Sistemas Informáticos
Curso 2004-2005

SISTEMA DE TESTING DE ASPECTOS CUANTITATIVOS

Itziar de Pablo Gutiérrez
Mercedes Donadiós Algarra
Miguel Sánchez Codoni

Dirigido por:
Prof. Manuel Núñez
Dpto. Sistemas Informáticos y Programación (SIP)

Facultad de Informática
Universidad Complutense de Madrid

Índice

El índice de la memoria aquí presentada está compuesto por los siguientes elementos:

1.- Resumen

El proyecto aquí presentado ha constado en la realización de una herramienta para el testeo de sistemas de aspectos cualitativos.

Se ha realizado dicha herramienta tanto para el testeo de especificaciones de propiedades funcionales, como de especificaciones de propiedades no funcionales, concretamente como ejemplo de propiedad no funcional se tiene el tiempo.

Este proyecto ha sido realizado en dos fases. La primera fase ha consistido en la implementación de un máquina de estados finita para la comprobación de la conformidad entre especificaciones e implementaciones.

Por otro lado, la segunda fase ha sido una ampliación de la primera para permitir, como se ha comentado anteriormente, el testeo de propiedades no funcionales a través del desarrollo de un sistema temporizado por un lado, y un sistemas con tiempos estocásticos por otro.

The project here mentioned has consisted on the creation of a tool for testing systems of qualitative features.

The tool has been implemented for testing specifications of functional properties, as well as for testing specifications of non functional properties. To be more precise, the time attribute is an example of these last type of properties.

This project has been carried out in two phases. The first one consists on the implementation of a finite states machine to check the conformance between specifications and implementations.

On the other hand, the second phase is based on the first one just mentioned in order to allow, the testing of non functional properties by means of the development of a timed system, firstly, and a system with stochastic times, finally.

2.- Palabras clave

Las palabras clave que más fácilmente pueden encontrarse o con mayor frecuencia a lo largo del desarrollo del proyecto son:

- **Especificación**
- **Implementación**
- **Derivación**
- **Test**
- **Conforme**

3.- Autorización

Los abajo firmantes, Dª Itziar de Pablo Gutiérrez, Mercedes Donadiós Algarra y Miguel Sánchez Codoni autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Itziar de Pablo Gutiérrez Mercedes Donadiós Algarra MiguelSánchez Codoni

4.- Desarrollo del proyecto. Contenido

Como ya se ha detallado en el resumen del proyecto, éste ha consistido en la implementación de un sistema para testing de aspectos cuantitativos.

El desarrollo del proyecto se ha dividido en dos fases.

- **Fase 1**: Propiedades funcionales
    - ✓ Máquina de Mealy (máquina de estados finita)
- **Fase 2**: Propiedades no funcionales
    - ✓ Sistema temporizado (introducción variable tiempo fijo)
    - ✓ Sistema con tiempos estocásticos (introducción de tiempo variable)

Ahora bien, a continuación vamos a pasara explicar de manera algo más detallada en qué han consistido dichas fases.

4.1.- Primera fase. Propiedades funcionales

4.1.1.- Máquina de Mealy. Generalidades

La **estructura general** de una máquina de Mealy, es la siguiente: Mealy = (Q, Ent, Sal, tran, res, q0), siendo:

- Q: conjunto de estados
- Ent: alfabeto de entrada
- Sal: alfabeto de salida
- tran: Q x Ent → Q, función de transición
- res: Q x Ent → Sal, función de respuesta
- q0 ∈ Q, estado inicial

y sigue la siguiente **semántica procedimental**:

- Estado inicial q0
- Estado $q \in Q$
- Recibe entrada: $e \in Ent$
- Emite salida: s = res(q, e)
- Transita nuevo estado: p = tran(q, e)

Ahora bien, para la máquina de Mealy que se ha implementado en este proyecto, y dado el objetivo del mismo, la característica fundamental que se ha seguido como máxima es:

"Una implementación se dice conforme con su especificación si la implementación no muestra outputs (salidas) no esperadas".

Ejemplo 1: Especificación genérica

Sea la siguiente especificación, tras la que la ejecución de una serie de transiciones, se obtiene que tras la introducción de una entrada i3, se espera obtener como respuesta una de entre un conjunto definido, tal y como se muestra a continuación:

Especificación: tras i1/o1, i2/o2, i3 → {o4, o5, o6}

La diferencia entre una implementación conforme y una implementación no conforme se muestra a continuación:

- Implementación conforme: tras i1/o1, i2/o2, i3 → {o4, o5, o6}
- Implementación no conforme: tras i1/o1, i2/o2, i3 → o7

Ejemplo 2: Máquina de Coca-Cola

Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:
Insertar moneda / oír ruido moneda caer, pulsar botón / iluminar botón, esperar / {sale coca cola, luz producto agotado}
Implementación conforme:
Insertar moneda / oír ruido moneda caer, pulsar botón / iluminar botón, esperar / sale coca cola

En cuanto al ejemplo de una **implementación no conforme**, tendríamos, que tras la ejecución de los pasos antes mencionados obtuviésemos, por ejemplo, una Fanta de naranja, tal y como se muestra a continuación:

4.1.2.- Proceso de testeo. Derivación de tests

Durante la implementación de este proyectos e ha desarrollado un proceso de testo para la consecución final del objetivo del mismo, la realización de testings sobre especificaciones.

Ahora bien, un test no es más que un conjunto de entradas y salidas, así como de un conjunto de instrucciones y de un conjunto de salidas esperadas.

Los tests tienen como origen una especificación, es decir, son derivados a partir de una especificación y su objetivo es la comprobación del a conformance del a implementación respecto del a especificación. Esta comprobación se lleva acabo mediante la aplicación de los tests derivados a las implementaciones.

Para realizar este proceso de derivación, se ha hecho uso, mediante su implementación, del siguiente algoritmo de derivación:

Input: M = (S, I,O, δ, sin).
Output: T = (S_, I,O, δ_, s0, SI, SO, SF , SP , ζ).

Initialization:
– S_ := {s0}, δ_ := SI := SO := SF := SP := ζ := □.
– Saux := {(sin, ξ0, s0)}.

Inductive Cases: Apply one of the following two possibilities until Saux = □.

1. If (sM, ξ, sT) □ Saux **then** perform the following steps:

(a) Saux := Saux − {(sM, ξ, sT)}.
(b) SP := SP □ {sT }; ζ(sT) := ξ.

2. If Saux = {(sM, ξ, sT)} is a unitary set **and** there exists i □ I such that out(sM, i) _= □ **then** perform the following steps:

(a) Saux := □.
(b) Choose i such that out(sM, i) _= □.
(c) Create a fresh state s_ /□ S_ and perform S_ := S_ □ {s_}.
(d) SI := SI □ {sT }; SO := SO □ {s_}; δ_ := δ_ □ {(sT , i, s_)}.
(e) For each o /□ out(sM, i) do

– Create a fresh state s__ /□ S_ and perform S_ := S_ □ {s__}.
– SF := SF □ {s__}; δ_ := δ_ □ {(s_, o, s__)}.

(f) For each o □ out(sM, i) do

– Create a fresh state s__ /□ S_ and perform S_ := S_ □ {s__}.
– δ_ := δ_ □ {(s_, o, s__)}.
– sM1:= after(sM, i, o).
– Let (sM, i, o, ξ1, sM1) □ δ. Saux := Saux □ {(sM1, ξ + ξ1, s__)}.

4.2.- Segunda fase. Propiedades no funcionales

4.2.1.- Sistema temporizado

Este sistema está basado en la máquina de Mealy expuesta anteriormente, es decir, en un sistema de propiedades funcionales.

Este sistema surge por tanto la necesidad de ampliación para la creación de sistemas basados en propiedades no funcionales como bien pueda ser, por ejemplo, el tiempo. En este caso se trata de un tiempo fijo introducido por el usuario.

Ahora bien, en este caso nos encontramos con que:

"Se dice que una implementación es conforme a una especificación no sólo si la implementación hace lo que debe, si no que lo hace en el tiempo que debe".

Por lo que nos encontramos con la obtención de éxitos ligados a tiempos y a la obtención de los valores esperados.

Ejemplo: Máquina de Coca-Cola

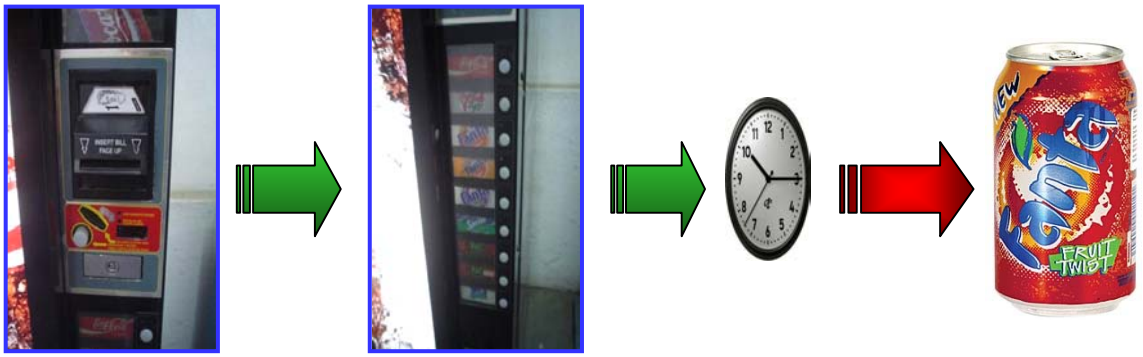
Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:
Insertar moneda / oír ruido moneda caer **(2 segundos)**, pulsar botón / iluminar botón **(1 segundo)**, esperar / {sale coca cola **(3 segundos)**, luz producto agotado **(1 segundo)**}

Implementación conforme:
Insertar moneda / oír ruido moneda caer, pulsar botón / iluminar botón, esperar / sale coca cola **(= 6 segundos)**

En cuanto al ejemplo de una **implementación no conforme**, tendríamos, que tras la ejecución de los pasos antes mencionados obtuviésemos, por ejemplo, una Fanta de naranja, o bien obtuviésemos la Coca-cola en un tiempo superior al esperado, tal y como se muestra a continuación:

2 segundos **1 segundo** **3 segundos** **> 6 segundos**



4.2.2.- Sistema con tiempos estocásticos

Este sistema está basado, como el caso anterior, en la máquina de Mealy expuesta previamente, es decir, en un sistema de propiedades funcionales.

Pero en este caso no se trata de tiempos fijos, sin o que se vana considerar tiempos variables definidos u obtenidos mediante la aplicación de distintas funciones de distribución.

Es decir, en este caso el atributo tiempo no es una propiedad que se pueda determinar a priori y de manera fija, sino que dependerá de una variable aleatoria pudiendo darse el caso (de hecho es lo normal) que un mismo test, tarde distintos tiempos.

De esta manera, para poder determinar si una implementación es conforme con la especificación es necesaria la aplicación reiterada de los tests sobre dicha implementación, así como la creación de contrastes de hipótesis que sirvan de base para la determinación del a conformance.

Ahora bien, en este caso nos encontramos con que:

"Se dice que una implementación es conforme a una especificación no sólo si la implementación hace lo que debe, sino si además:
El contraste de hipótesis determina que los resultados obtenidos son lo suficientemente similares a los esperados por la función de distribución".

Ejemplo: Máquina de Coca-Cola

Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:
Insertar moneda / oír ruido moneda caer **(x segundos)**, pulsar botón / iluminar botón **(y segundo)**, esperar / {sale coca cola **(z1 segundos)**, luz producto agotado **(z2 segundo)**}

Y en cuanto a los tests, tenemos:

El grado de confianza es un factor que puede ser determinado por el usuario.

5.- Estructura del a implementación

La herramienta que se ha implementado como proyecto y que aquí se expone, es una aplicación Java con su correspondiente estructura de clases.

En este apartado se pretende reflejar dicha estructura con el objetivo de que posibles programadores que deseen ampliar el sistema dispongan de la información detallada para tal fin.

No obstante, es importante resaltar que no se pretende exponer aquí el código implementado o la explicación del mismo puesto que para tal finalidad está disponible el Javadoc generado por la aplicación.

5.1.- Estructura general

La estructura general del a aplicación consta de una estructura de paquetes como a continuación se muestra y con la siguiente información:

- **Paquete Maquina_Mealy.Estado**: contiene la implementación referida a los estados que constituyen las especificaciones.

- **Paquete Maquina_Mealy.Functions**: contiene la implementación referida a las distintas funciones de distribución consideradas en la aplicación usadas para los sistemas con tiempos estocásticos.

- **Paquete Maquina_Mealy.GUI**: contiene la implementación de todas las interfaces de usuario utilizadas por la aplicación.

- **Paquete Maquina_Mealy.Mealy**: contiene el código tanto de las especificaciones creada como de sus correspondientes implementaciones.

- **Paquete Maquina_Mealy.Tests**: contiene la implementación referida a los tests resultantes de la derivación de las especificaciones.

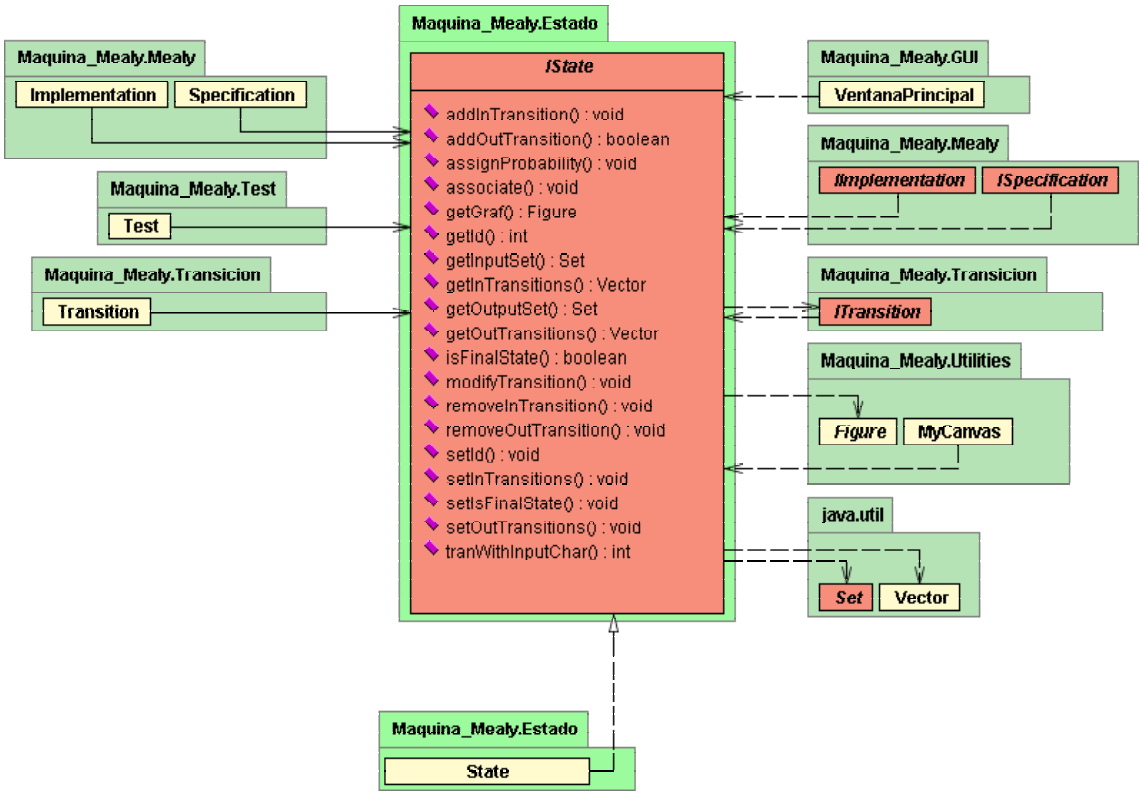
- **Paquete Maquina_Mealy.Transicion**: contiene la implementación referida a las transiciones que constituyen las especificaciones.

- **Paquete Maquina_Mealy.Utilities**: contiene la implementación de distintas utilidades necesarias para aportar la funcionalidad adecuada a la herramienta desarrollada.

5.1.1.- Paquete Maquina_Mealy.Estado

Este paquete está constituido por un interfaz "IState.java" y una clase "State.java" que lo implementa. Se muestra a continuación los diagramas UML correspondientes a ambos elementos:

Interfaz "IState.java"

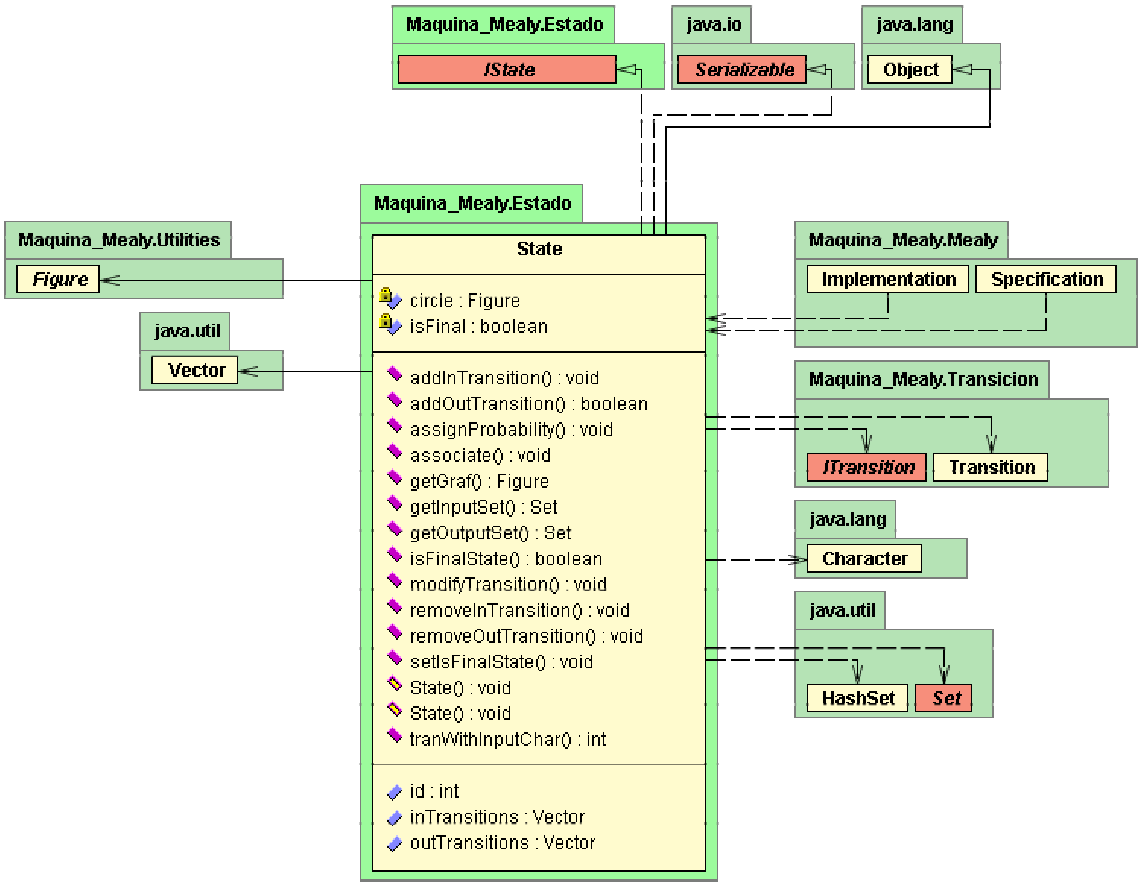


En esta interfaz están declarados todos los métodos que posteriormente serán implementados en la correspondiente clase State.java.

La principal función de esta interfaz es la declaración de los métodos relacionados no sólo con la definición de las características propias de un estado, tales como, la declaración de los accesores y mutadores de sus atributos, si no de aquellos métodos requeridos para la definición de las transiciones asociadas al mismo.

Clase "State.java"

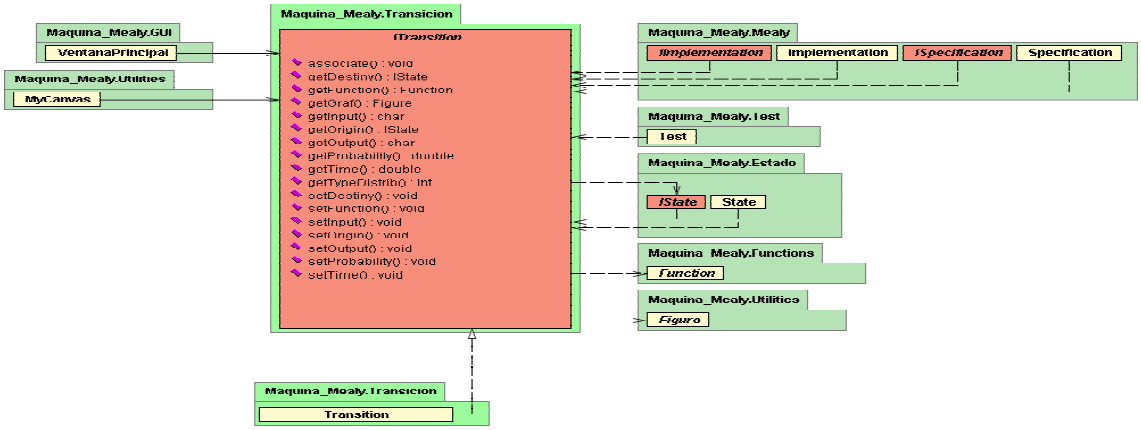
Implementa los métodos declarado sen su interfaz.



5.1.2.- Paquete Maquina_Mealy.Transicion

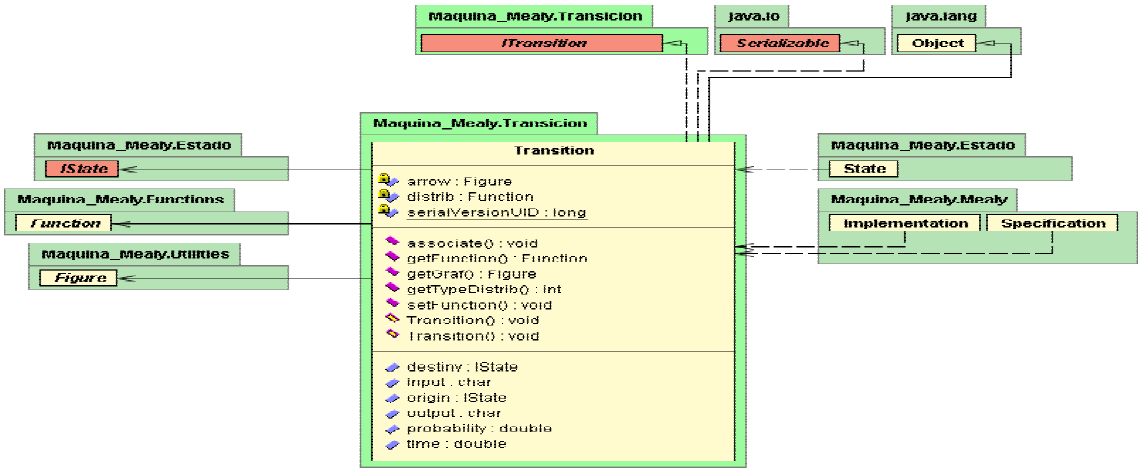
Este paquete está constituido por un interfaz "ITransition.java" y una clase "Transition.java" que lo implementa. Se muestra a continuación los diagramas UML correspondientes a ambos elementos:

Interfaz "ITransition.java"



En este caso se tienen definidos los métodos correspondientes a la definición de transiciones y a las características de éstas, así como de su integración y comunicación con el resto de la especificación.

Clase "Transition.java"



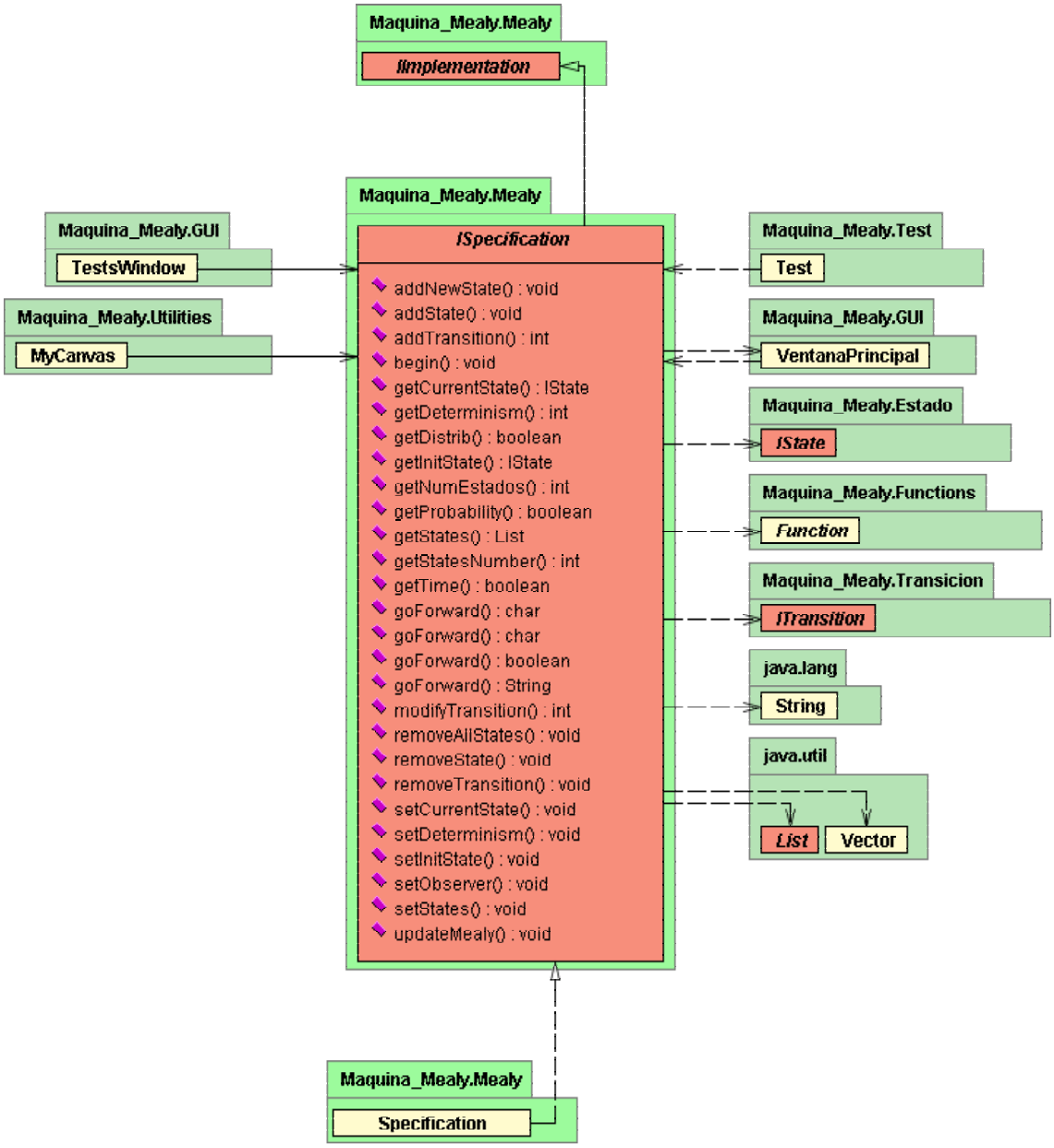
Implementa los métodos declarados en su correspondiente interfaz.

5.1.3.- Paquete Maquina_Mealy.Mealy

Este paquete está constituido por dos interfaces "ISpecification.java" y "IImplementation.java" así como las clases que las implementan, "Specification.java" y "Implementation.java" respectivamente.

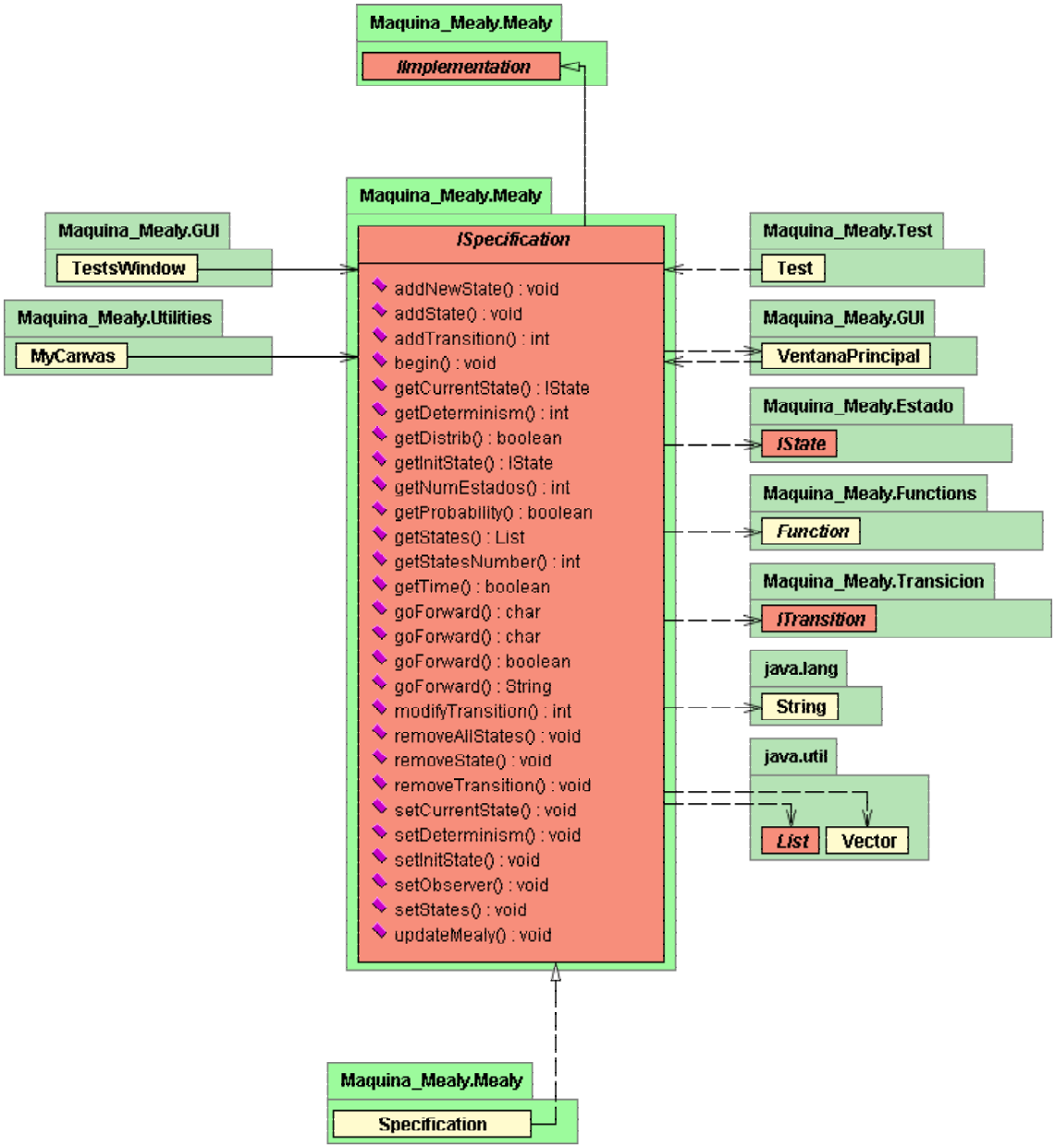
Se muestra a continuación los diagramas UML correspondientes a todos los elementos:

Interfaz "ISpecification.java"



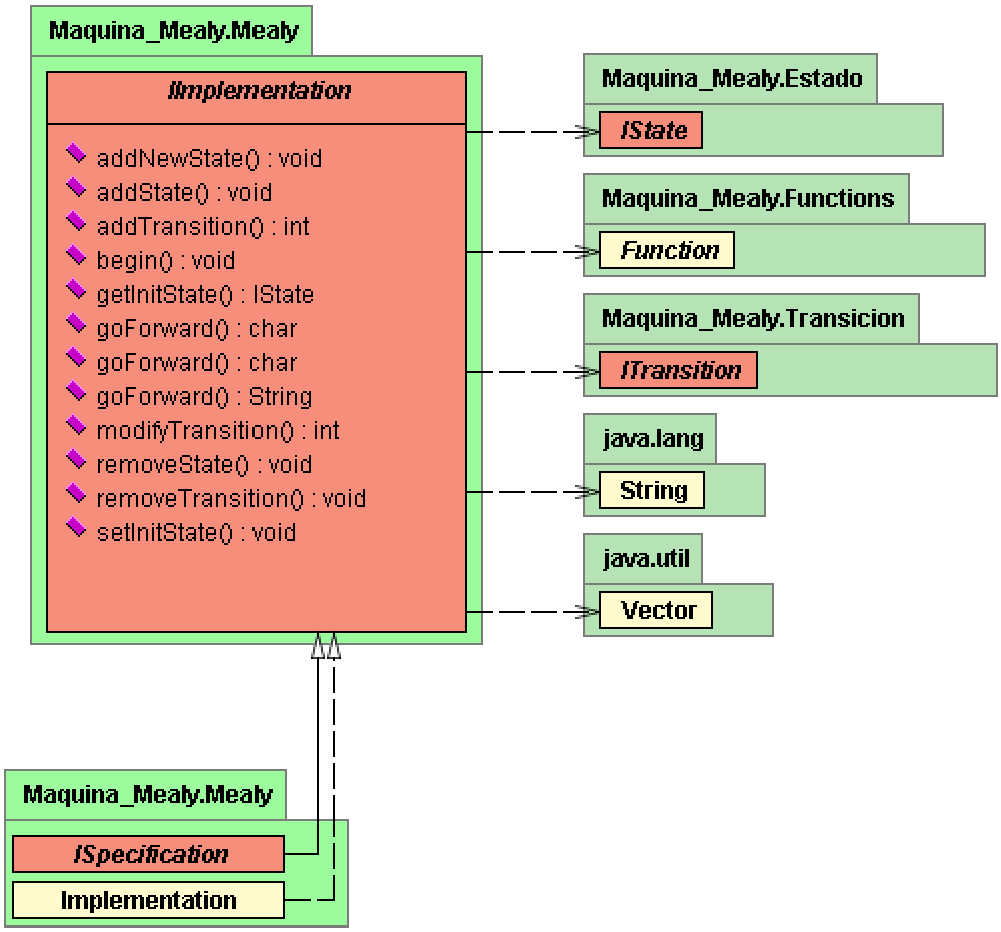
En esta interfaz se encuentran declarados todos los métodos necesarios para la creación del as especificaciones y del as características que soporta y que hayan sido previamente declarada sen función del tipo de especificación a diseñar.

Clase “Specification.java”



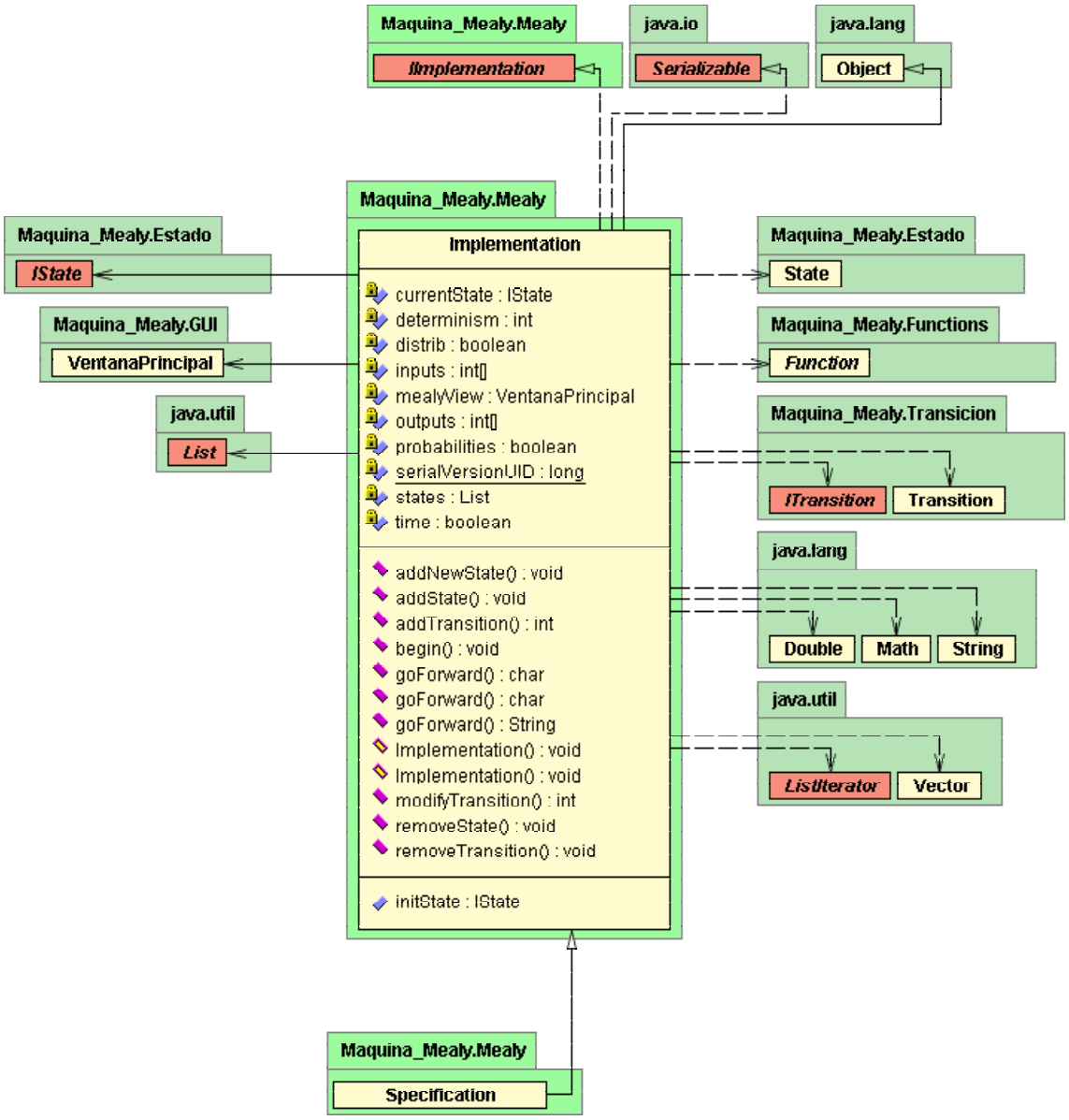
Implementa los métodos declarados en su correspondiente interfaz.

Interfaz “IImplementation.java”



En esta interfaz representa la implementación que se ha de comprobar si es conforme con la especificación anteriormente creada. De hecho, los métodos aquí implementados son los mismos que los implementados en la interfaz ISpecification salvo por las restricciones lógicas.

Clase “Implementation.java”



Implementa los métodos declarados en su correspondiente interfaz.

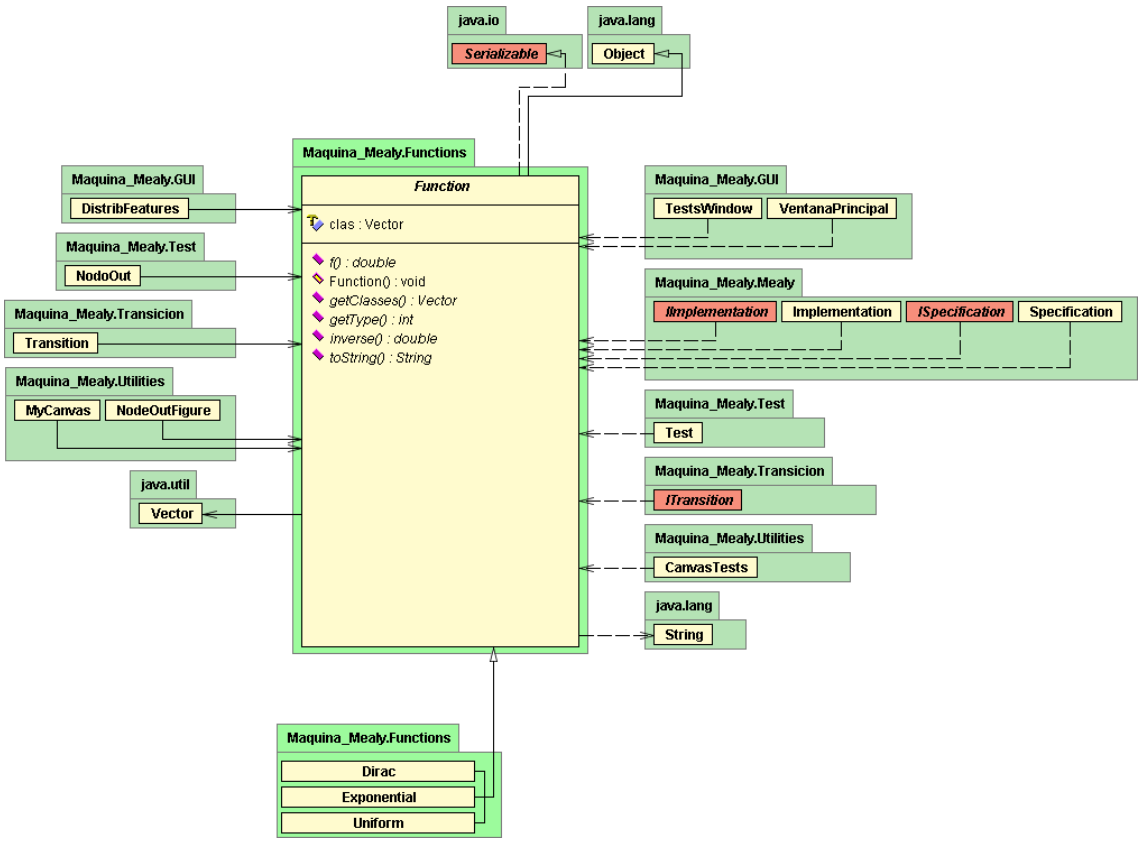
5.1.4.- Paquete Maquina_Mealy.Function

Este paquete está constituido por cuatro clases Java que implementan las funciones de distribución desarrolladas para la creación de sistemas con tiempos estocásticos. Es un de los paquetes con mayor interés en cuanto a posibles ampliaciones futuras, pues será aquí donde nuevas funciones de distribución deberían ser implementadas debido a la existencia, como se mostrará a continuación de una clase perteneciente a este paquete que servirá como plantillas para dichas adiciones.

Estas clases representan las funciones necesarias para aportar a las especificaciones de una propiedad no funcional como es el tiempo de una manera variable, es decir, dicho tiempo se calculará en función de la aplicación de una función u otra, siendo tres, el número de funciones de distribución que se han implementado.

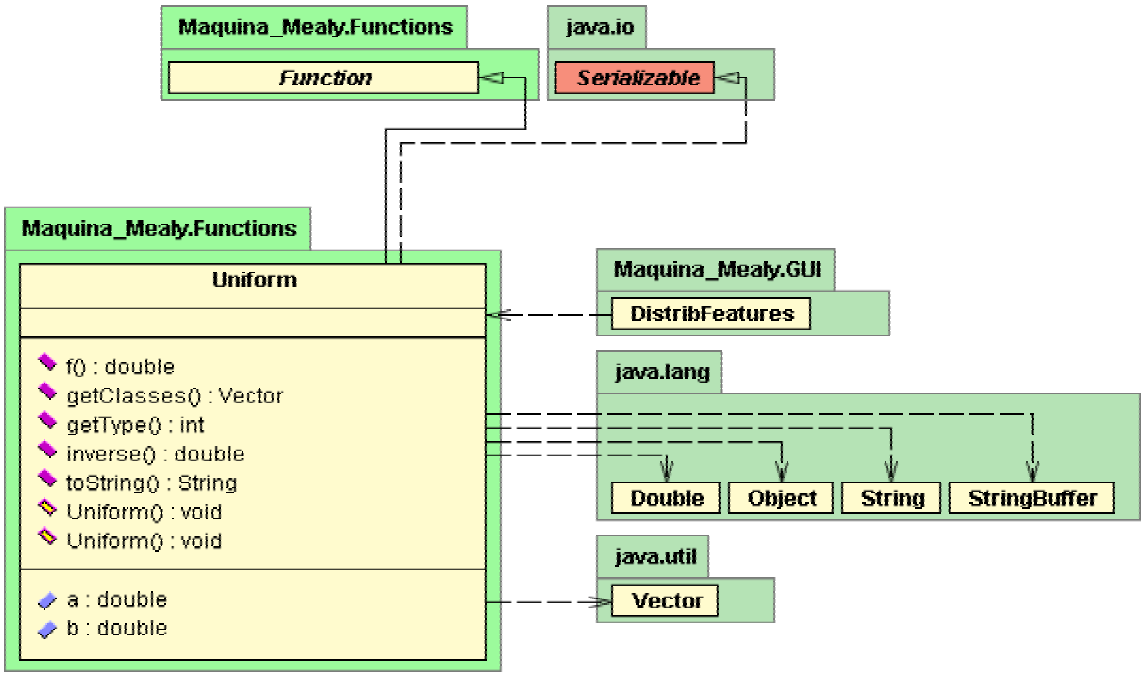
Ahora bien, mostramos a continuación los diagramas UML del as clases pertenecientes a este paquete.

Clase "Function.java"



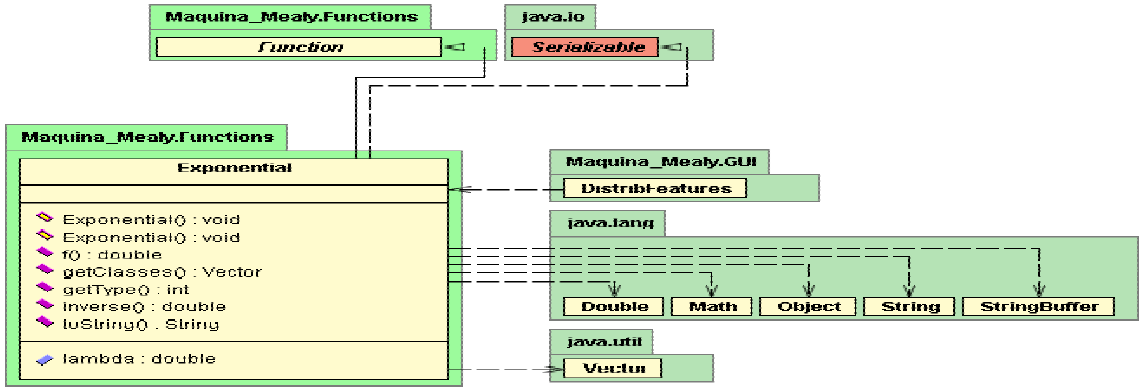
Esta es una clase abstracta que sirve como plantilla para la creación de clases que deriven de ella y que representen distintas funciones de distribución que puedan ser aplicadas a especificaciones con propiedades no funcionales como ocurre en los sistemas con tiempos estocásticos que aquí se desarrollan.

Clase "Uniform.java"



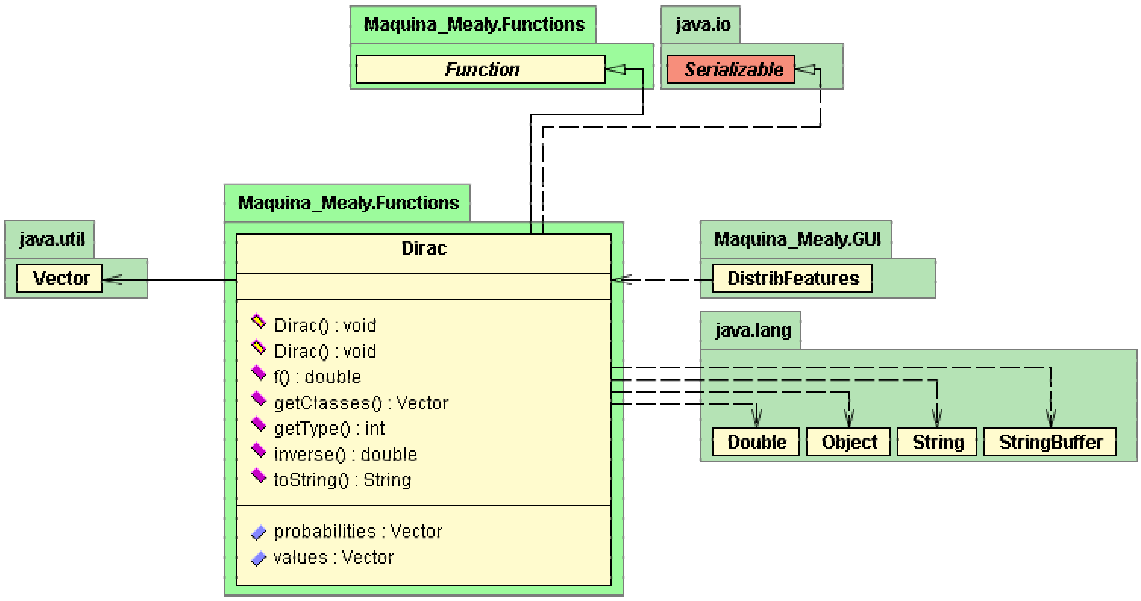
Clase que representa a la función uniforme.

Clase "Exponential.java"



Esta clase representa la función de distribución exponencial.

Clase “Dirac.java”



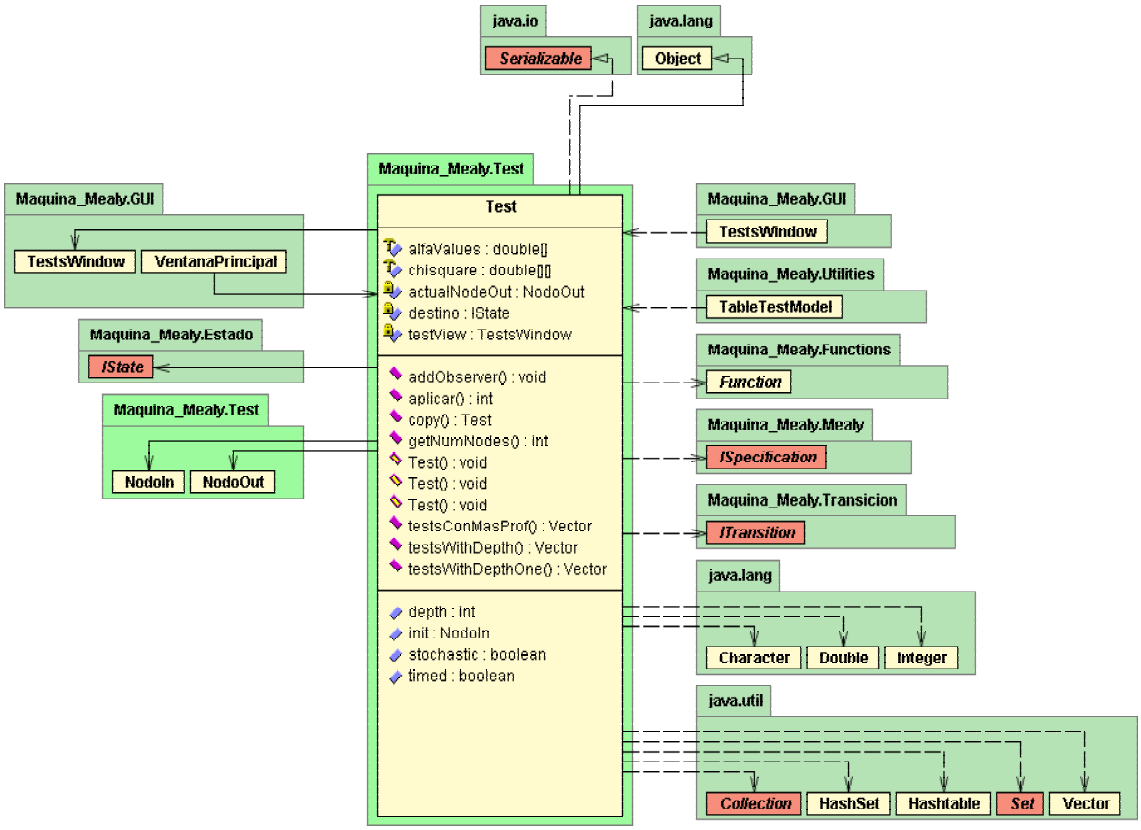
Esta clase representa la función de distribución Dirac, o también conocida como función de saltos.

5.1.5.- Paquete Maquina_Mealy.Tests

Este paquete está constituido por tres clases mediante las cuales se ha implementado los tests resultantes de la derivación de las especificaciones.

A continuación se muestra los diagramas UML correspondientes a dichas clases.

Clase "Test.java"



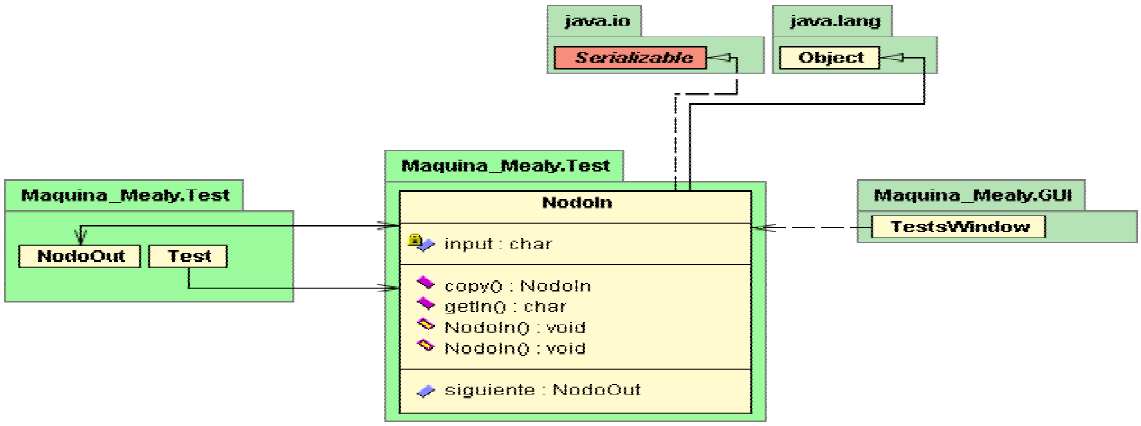
En esta clase se encuentran los métodos necesarios para la representación de los tests resultado de la derivación. Es en esta clase donde se realiza la derivación, es decir, donde se lleva acabo el algoritmo de derivación seguido, para que partiendo de una especificación previamente creada se construyan los tests que serán aplicados a continuación a la implementación correspondiente.

Este algoritmo parte del método "*testWithDepthOne*", y mediante el método "*testWithDepth*" y el uso del anterior así como de un proceso iterativo, se obtienen

todos los tests obtenidos de la especificación y con la profundidad marcada por el usuario, como se indica en el manual de usuario.

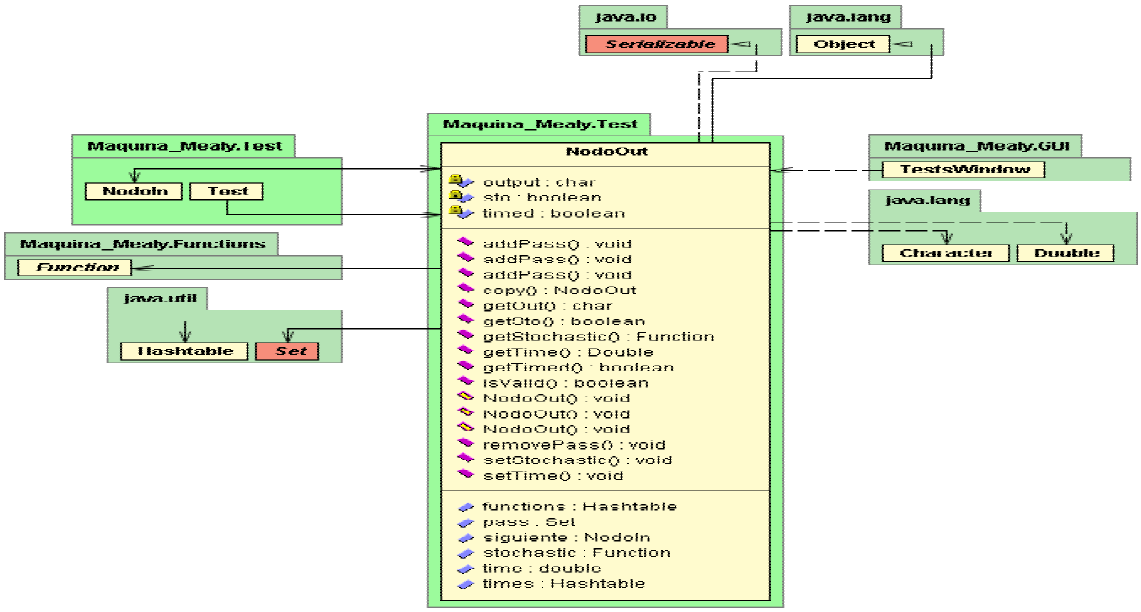
Ahora bien, hay que tener en cuenta que un test consta tanto de nodos de entrada (representando entradas de transiciones) como por nodos de salida (representando salidas de transiciones y el conjunto de salidas rechazadas).

Clase "NodoIn.java"



Esta clase representa un nodo de entrada del test, es decir, una entrada de una transición del a especificación.

Clase "NodoOut.java"

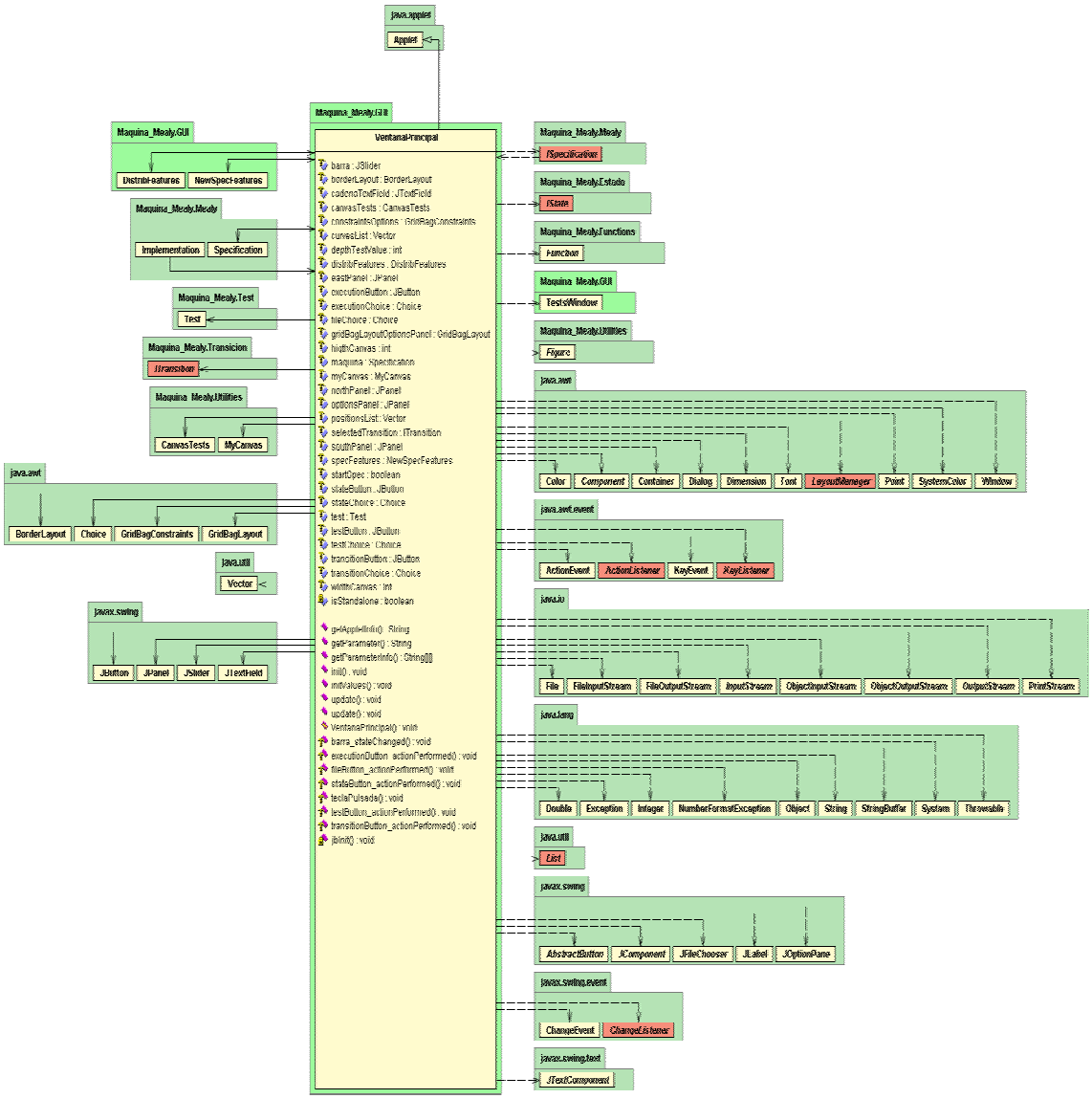


5.1.6.- Paquete Maquina_Mealy.GUI

Este paquete está constituido por todas aquellas clases que representan interfaces gráficas de usuario para la ínteractuación con el mismo.

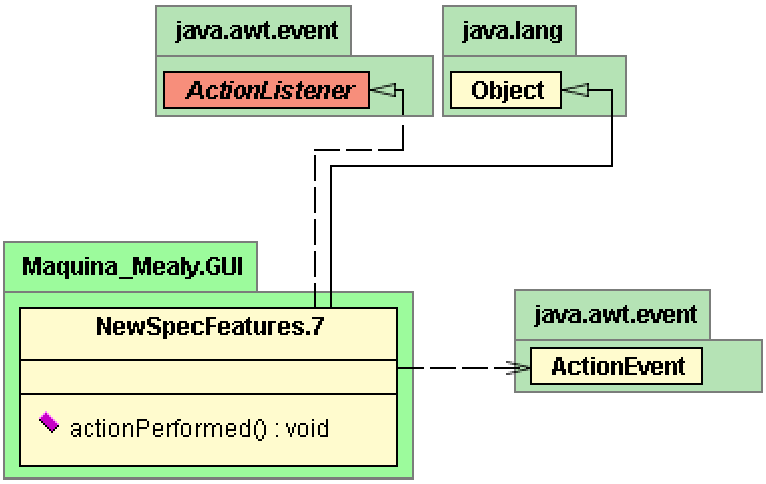
Se muestran a continuación los diagramas UML asociados a dichas clases.

Clase "VentanaPrincipal.java"



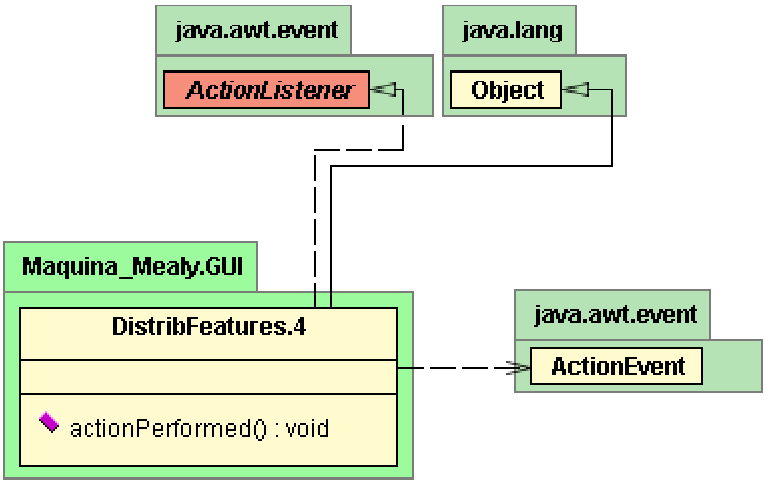
Es la interfaz principal del a aplicación ye l punto de entrada a la misma. Es desde esta ventana desde donde se tiene acceso a todas las opciones que la herramienta soporta. Estas opciones están disponibles a través de elementos JChoice.

Clase “NewSpecFeatures.java”



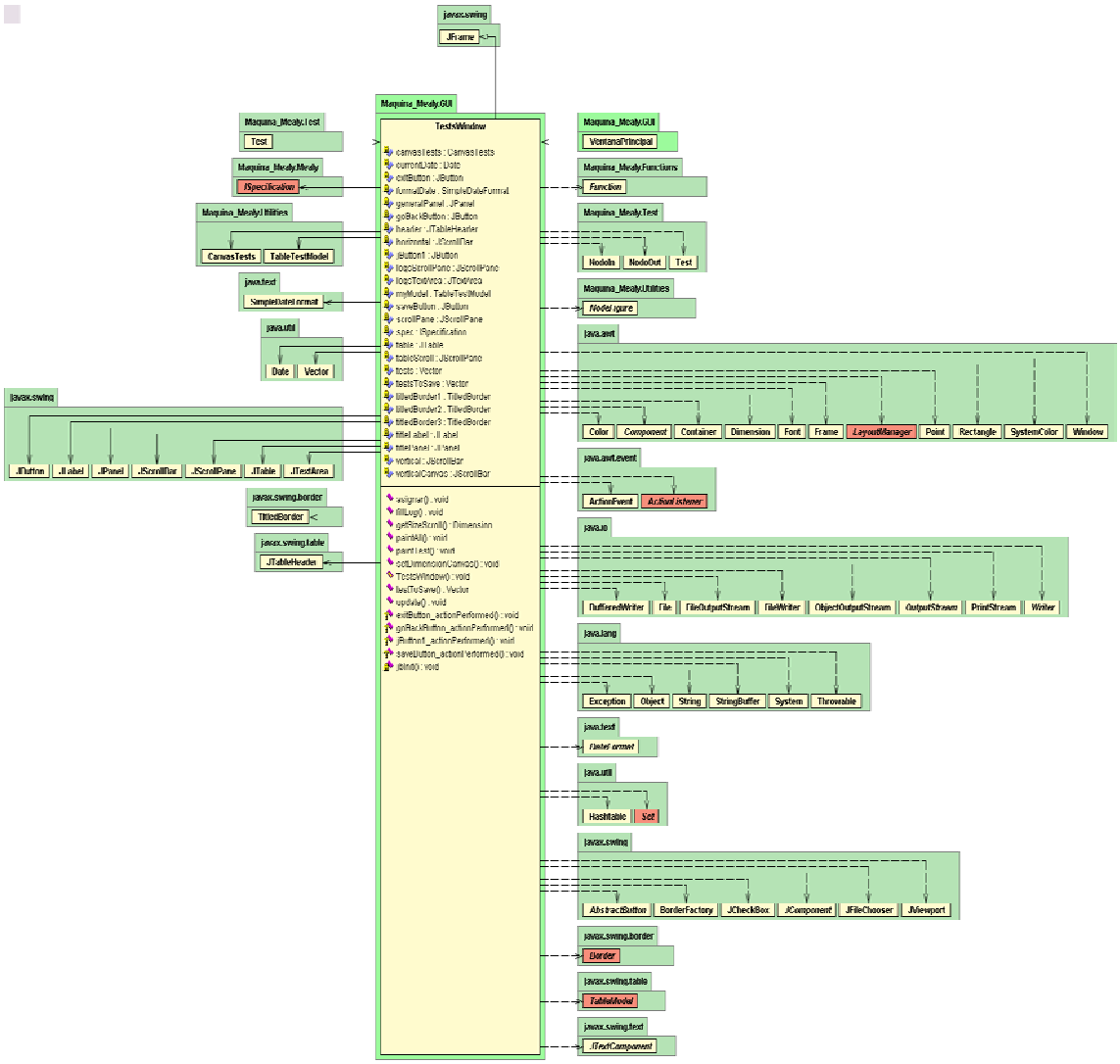
Esta clase es la que implementa la interfaz para la introducción por parte del usuario del tipo de especificación que va a ser creada así como las características de la misma.

Clase “DistribFeatures.java”



Esta es la clase en la que se implementa la interfaz para que el usuario vaya seleccionando en cada transición que cree, la función de distribución que va a utilizar para el cálculo del tiempo de dicha transición. Será utilizada en aquellos casos sen los que se haya decidido diseñar un sistema con tiempos estocásticos.

Clase “TestsWindow.java”



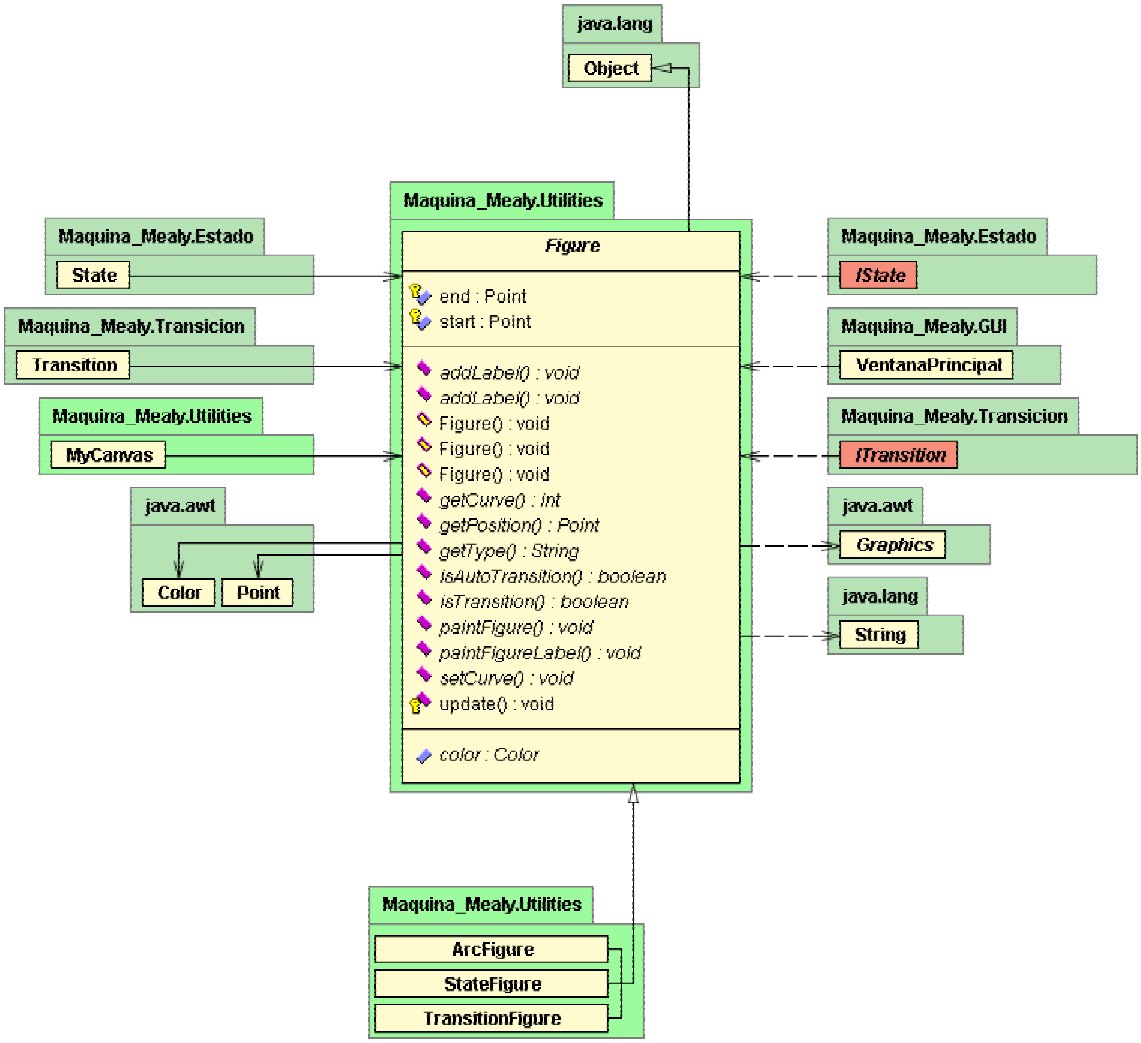
En esta clase está implementada la interfaz que representa la visualización de los tests generados tras la derivación. Es en esta interfaz, donde se ofrece al usuario, la posibilidad al usuario de aplicar los tests a la implementación para la comprobación de la conformance de ésta última con la especificación creada.

5.1.7.- Paquete Maquina_Mealy.Utilities

Este paquete está integrado por un aserie de clases necesarias para la representación gráfica de los elementos de los que consta la herramienta, es decir, para la representación gráfica tanto de las especificaciones como de los tests. Así mismo se ha implementado en este paquete clases auxiliares utilizadas por la herramienta para su correcto funcionamiento.

Se muestran a continuación los diagramas UML asociados a dichas clases.

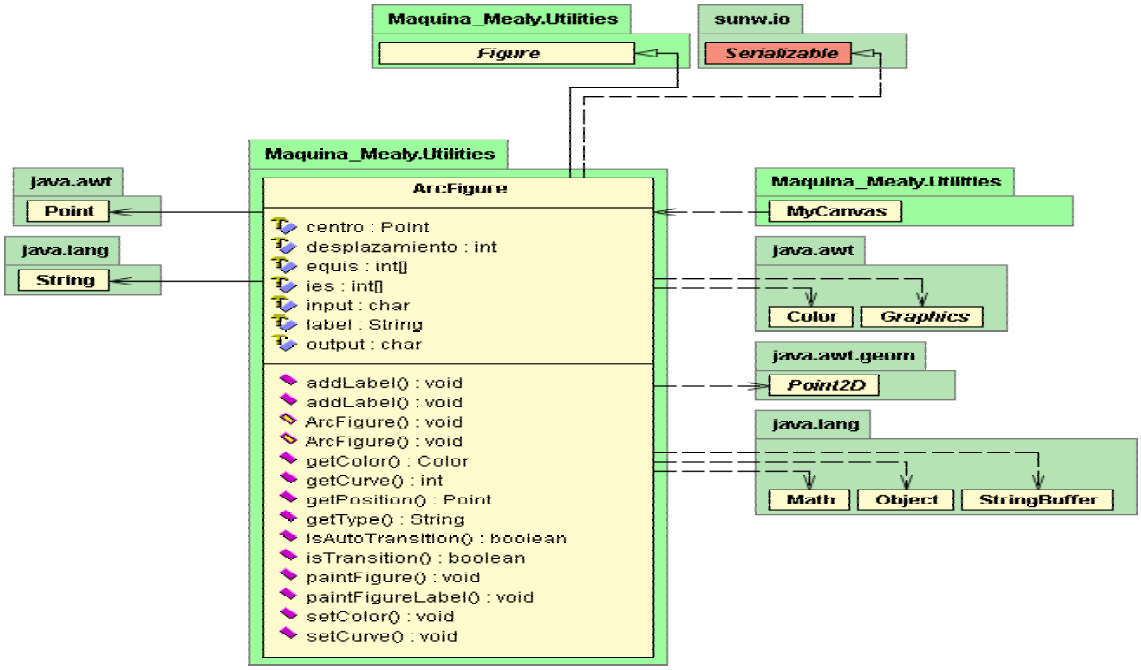
Clase "Figure.java"



Esta clase es una clase abstracta que sirve como plantilla para los distintos tipos de figuras que se van a dibujar en la aplicación para la creación de especificaciones,

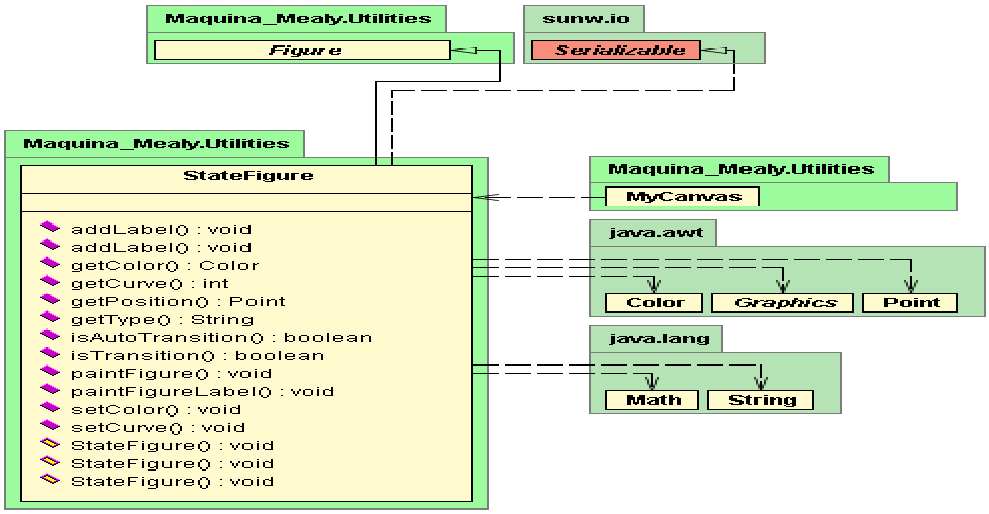
concretamente, para la creación de figuras que representen: estados, transiciones y auto transiciones.

Clase "ArcFigure.java"



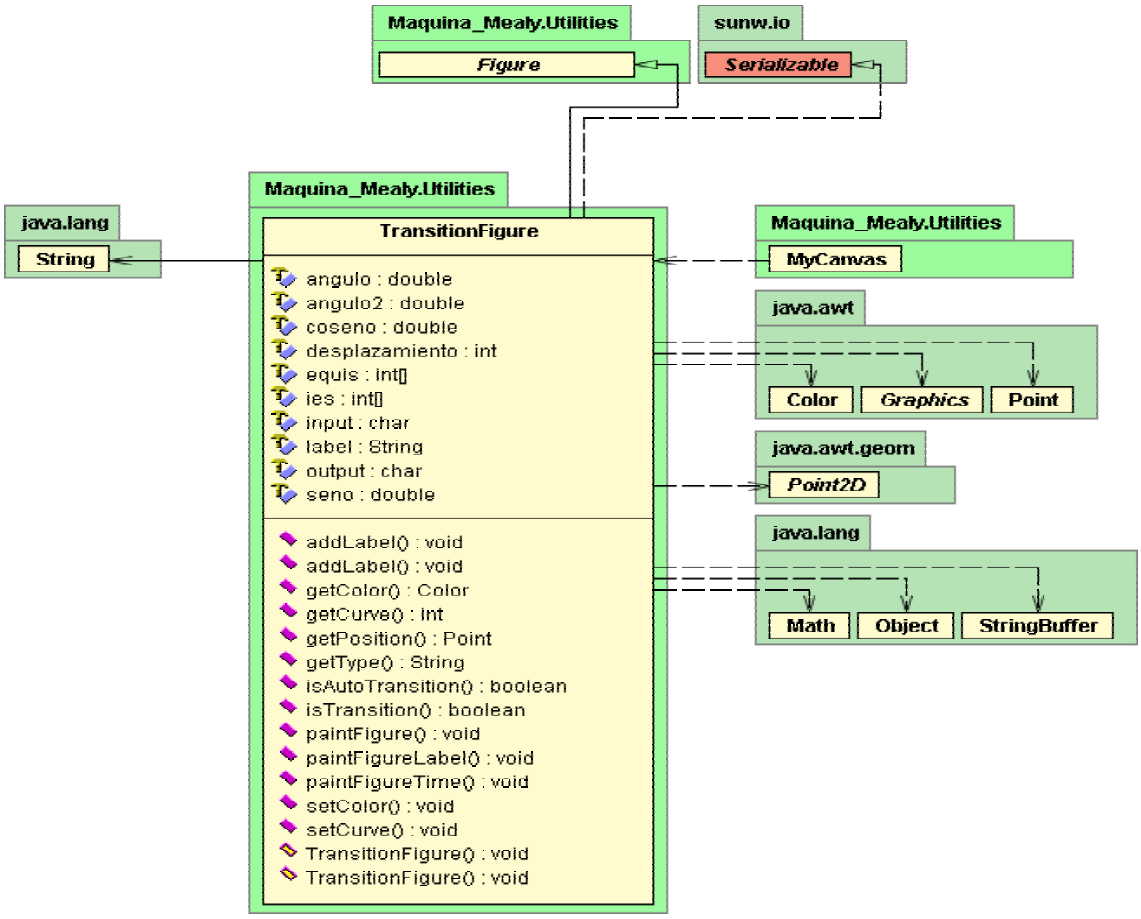
Clase implementada para representar gráficamente una auto transición.

Clase "StateFigure.java"



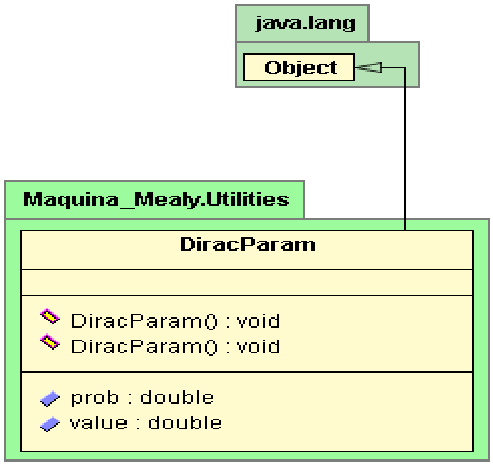
Clase implementada para representar gráficamente un estado.

Clase “TransitionFigure.java”



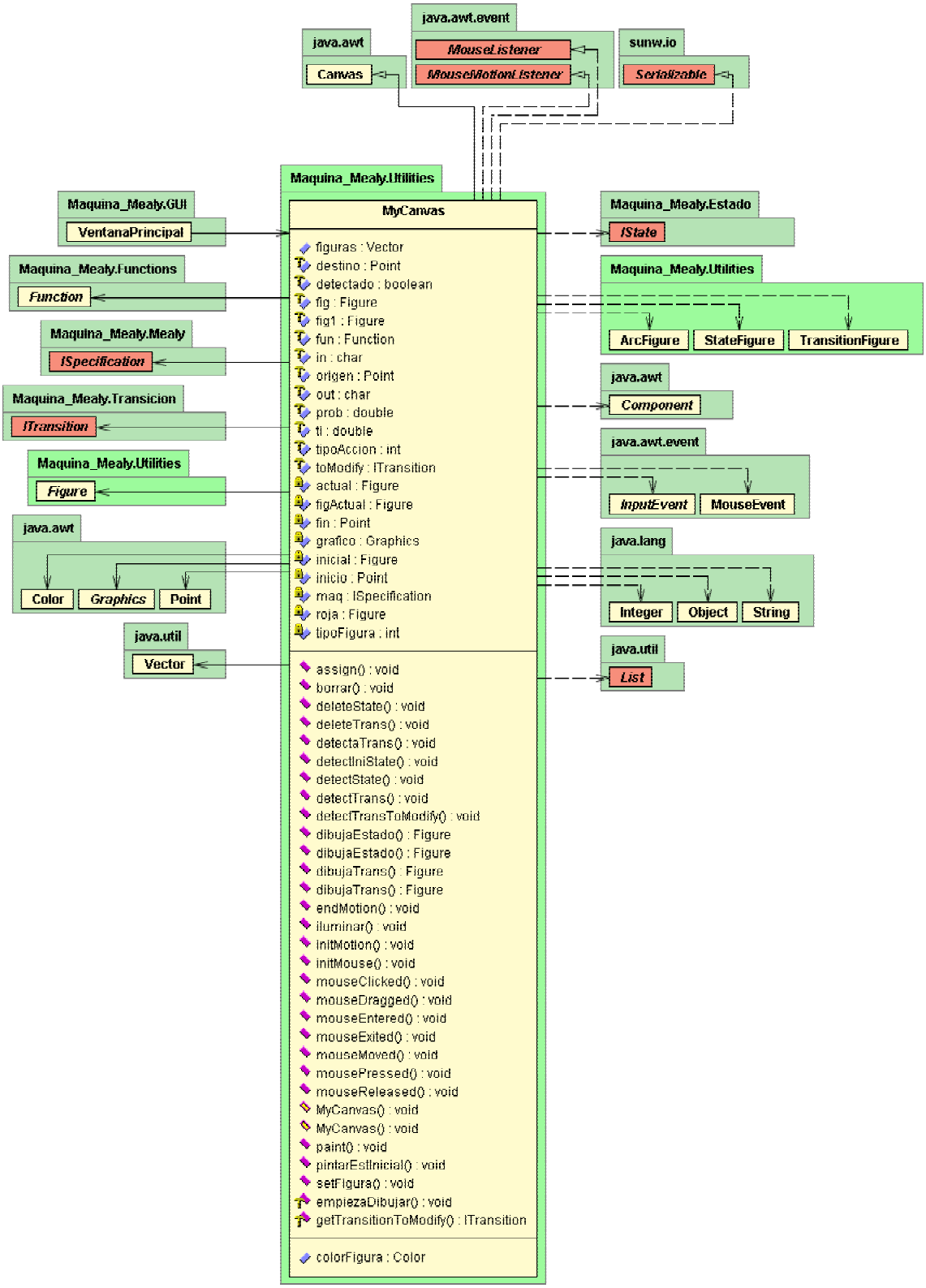
Clase implementada para representar gráficamente una transición.

Clase “DiracParam.java”



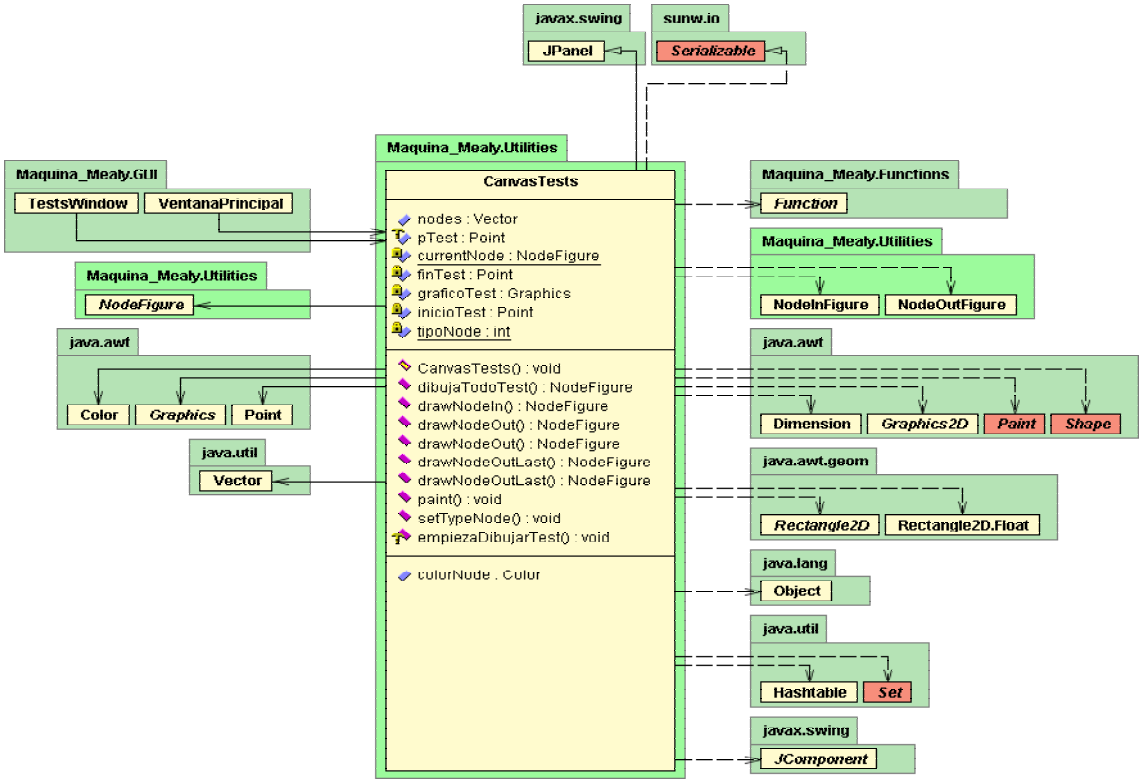
Clase implementada para la manipulación de los parámetros de la función de distribución Dirac.

Clase "MyCanvas.java"



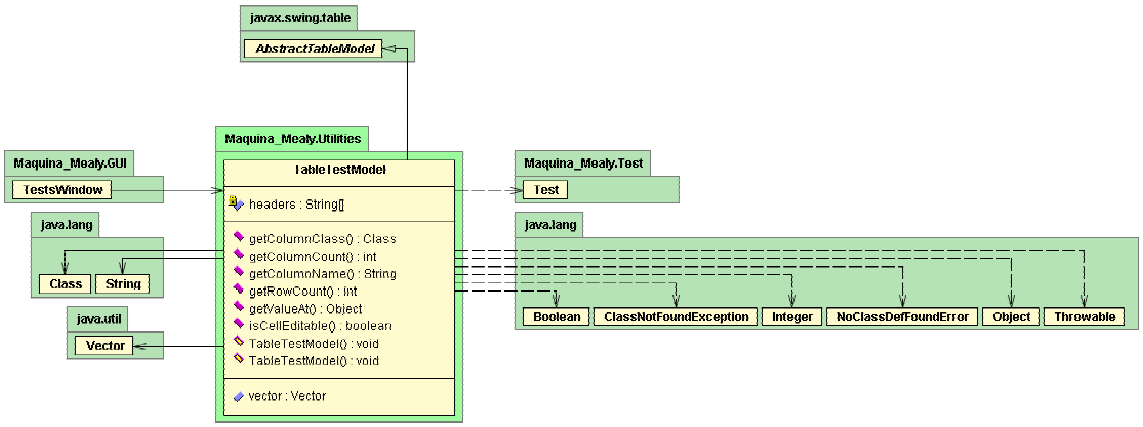
Esta clase implementa todos los métodos necesarios para la representación gráfica en el panel de edición al uso de la ventana principal, de las especificaciones que se creen.

Clase "CanvasTests.java"



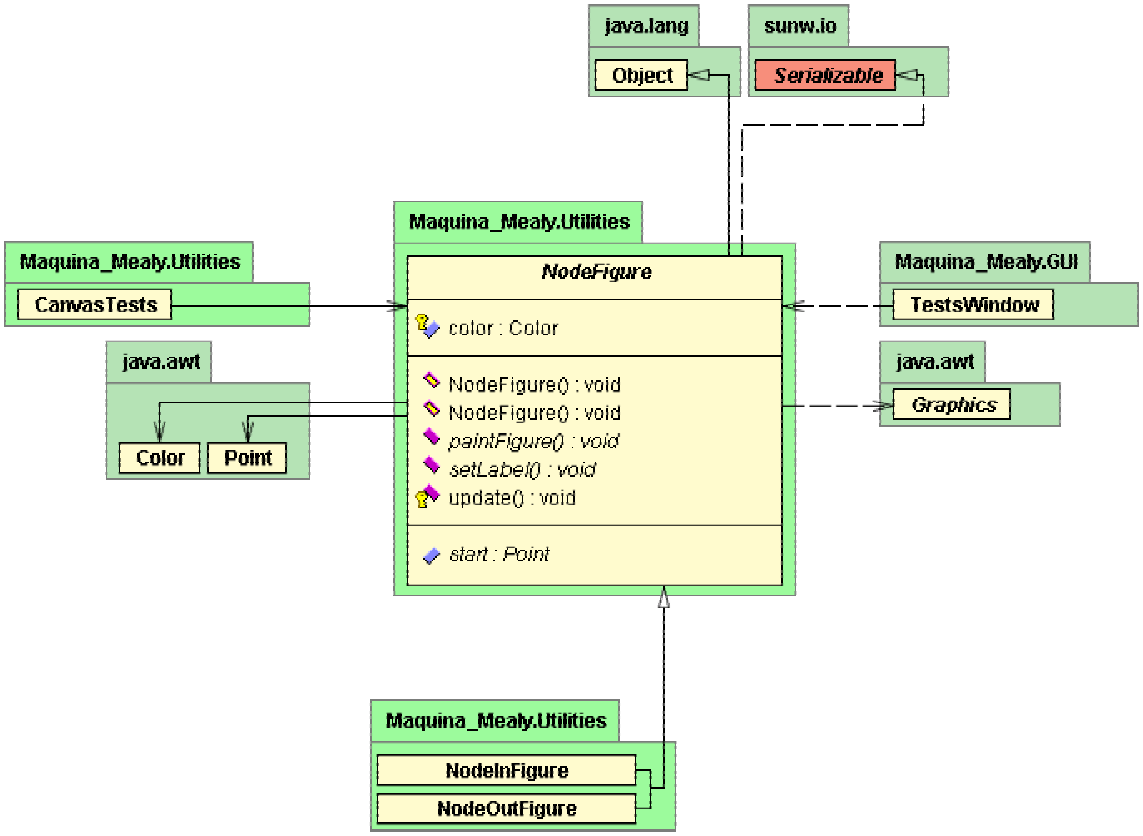
Esta clase implementa todos los métodos necesarios para la representación gráfica en el panel de edición al uso de la ventana asociada a la visualización de tests, de los tests que hayan sido derivados.

Clase “TableTestModel.java”



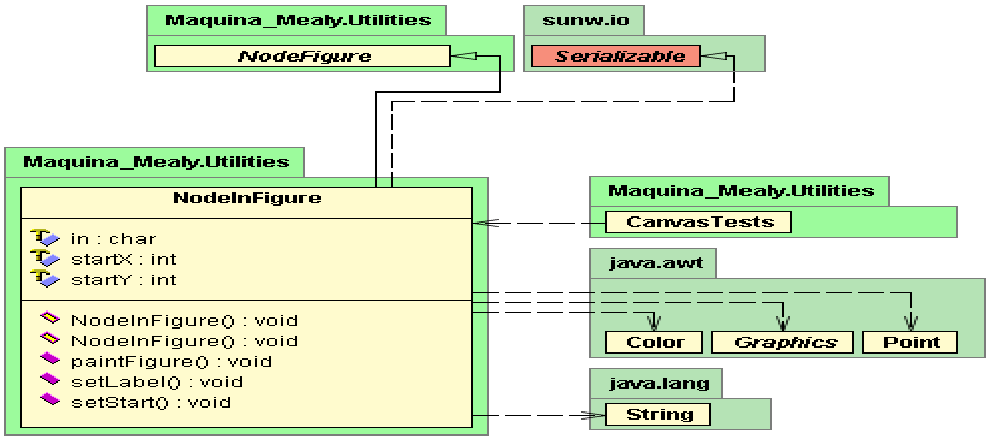
Esta clase implementa el modelo de tabla utilizado en la interfaz de visualización de tests, y en la que se almacenan algunas del as características de los tests que se han obtenido como resultado del a derivación.

Clase “NodeFigure.java”



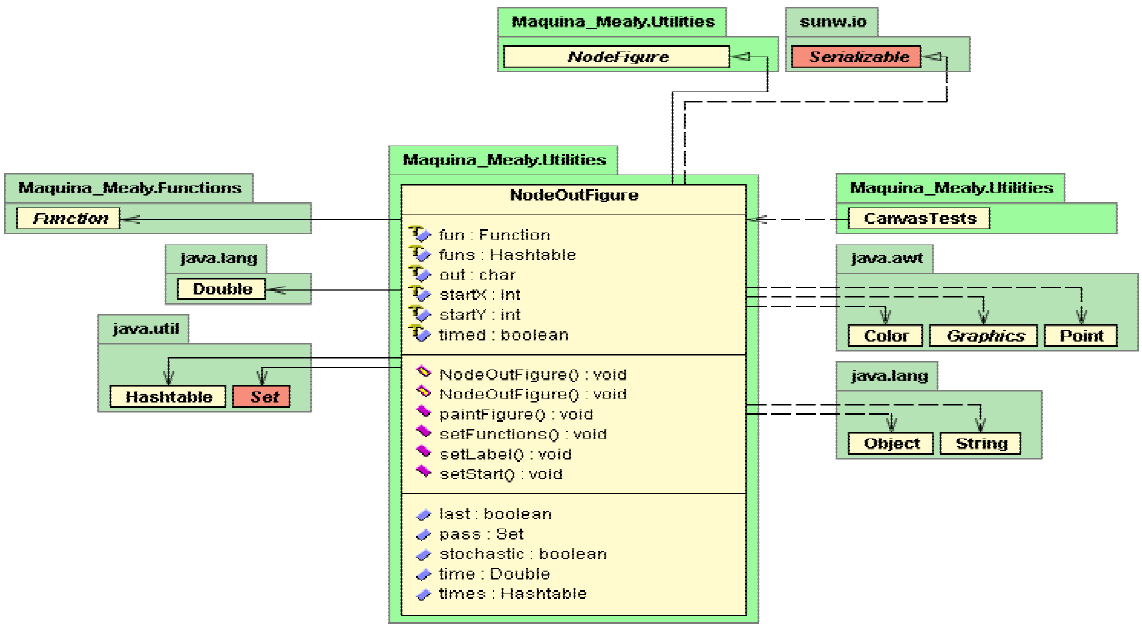
Esta clase es una clase abstracta que sirve como plantilla para los distintos tipos de figuras que se van a dibujar en la aplicación para la visualización de los tests, concretamente, para la creación de figuras que representen: nodos de entrada y nodos de salida.

Clase "NodeInFigure.java"



Clase implementada para representar gráficamente un nodo de entrada de un test.

Clase "NodeOutFigure.java"



Clase implementada para representar gráficamente un nodo de salida en un test.

6.- Manual de Usuario

6.1.- Introducción

A lo largo de este manual se exponen las distintas opciones de las que consta la herramienta aquí presentada, así como los pasos necesarios a seguir para la ejecución de las mismas.

Pretende ser por tanto, una guía que permita a cualquier usuario beneficiarse de las características proporcionadas por el sistema implementado de manera sencilla, y que el manejo de la aplicación no resulte un impedimimento para lograr los objetivos buscados: testear propiedades cuantitativas.

No obstante, se ha realizado una implementación y se ha creado una estructura de clases pensando en el usuario final de la aplicación, de tal forma que de una manera rápida e intuitiva pueda manejar la herramienta y descubrir en poco tiempo y sin esfuerzo los beneficios del sistema.

6.2.- Requisitos previos necesarios

Para la ejecución de la herramienta y el correcto funcionamiento de la misma es necesario disponer previamente de:

- Una Máquina Virtual de Java (versión jdk1.3.1 o superior. Se recomienda versión jdk1.4.2).

- Eclipse 7.0: en caso de no disponer de este software puede realizarse la ejecución de la herramienta a través de un .bat, con el siguiente contenido:

  - ✓ Path a la MVJ.

  - ✓ Llamada al método principal de la aplicación.

6.3.- Instalación y configuración

La implementación de la aplicación así como sus componentes se encuentran almacenados en un archivo llamado "TestingTool.zip", cuyo contenido es el siguiente:

- Implementación: contiene la implementación de la aplicación.

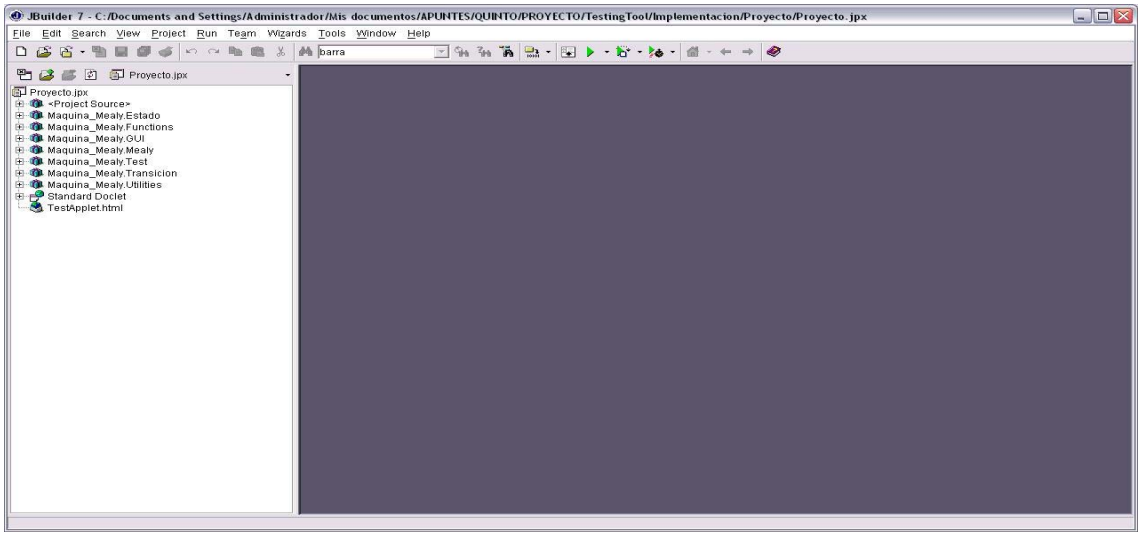
  - ✓ Proyecto: contiene un proyecto realizado con JBuilder, con la siguiente estructura:
    - Classes: contiene las clases compiladas de la aplicación.
    - Src: contiene los fuentes (ficheros .java) de la aplicación.
    - Proyecto.jpx: permite la apertura con JBuilder de la aplicación.

- Javadoc: contiene el Javadoc generado por la aplicación.

Ahora bien, para ejecutar la aplicación, y una vez que se dispone de los requisitos previos anteriormente mencionados, será necesario seguir los siguientes pasos:

- Descomprimir el archivo "TestingTool.zip".

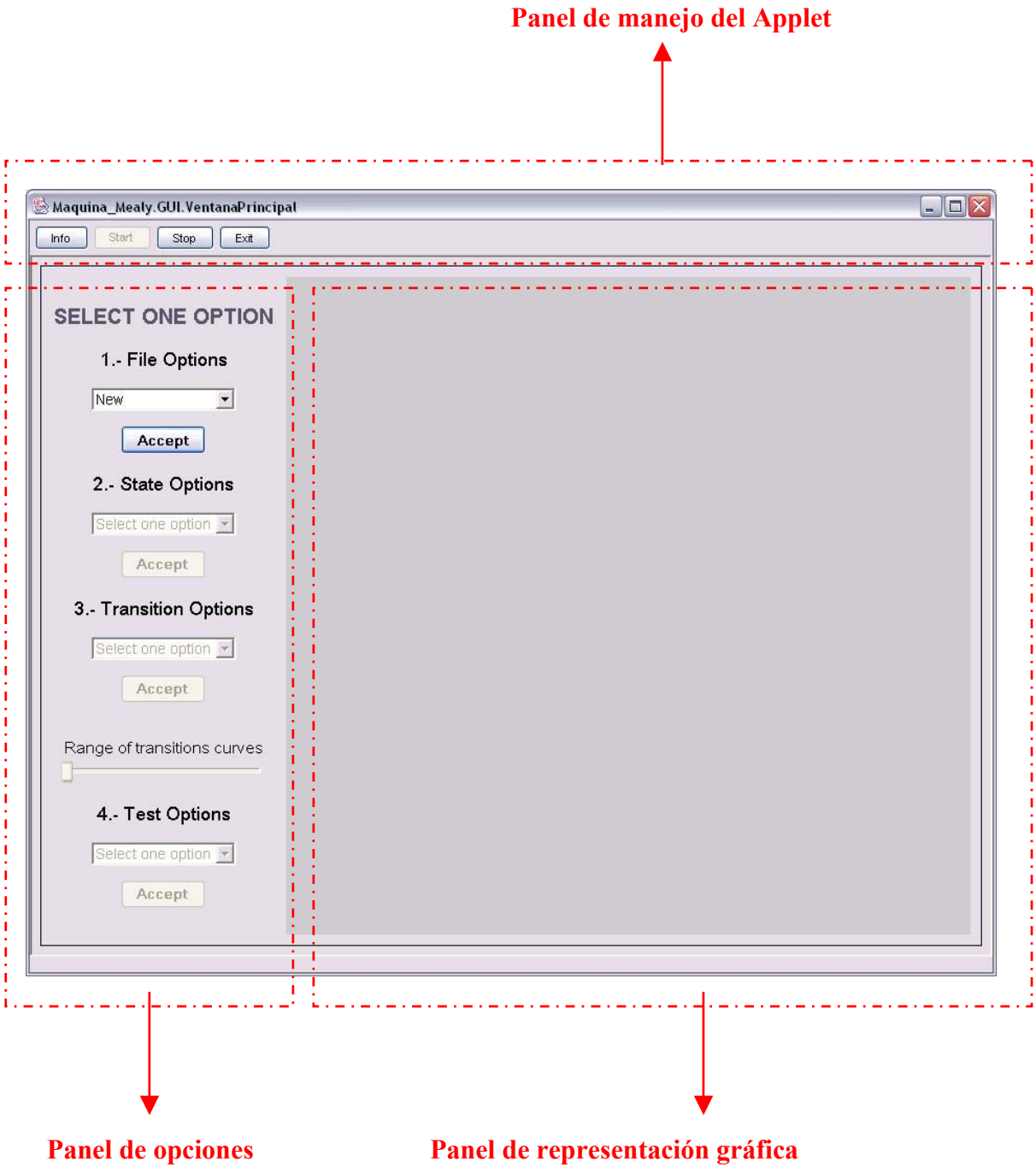
- Abrir el fichero TestingTool / Implementación/ Proyecto / Proyecto.jpx con un doble click sobre el mismo. Se accederá entonces al IDE de JBuilder, mostrándose la siguiente ventana:



6.4.- Ejecución de la herramienta. Funcionamiento

A continuación se va a detallar el funcionamiento de la herramienta y cómo navegar por las distintas opciones que ésta proporciona.

En primer lugar será necesario lanzar la aplicación. Para ello, una vez abierto el proyecto con JBuilder, basta pulsar sobre el icono , con lo que se accederá a la ventana principal de la aplicación, que se muestra a continuación:



6.4.1.- Crear una especificación

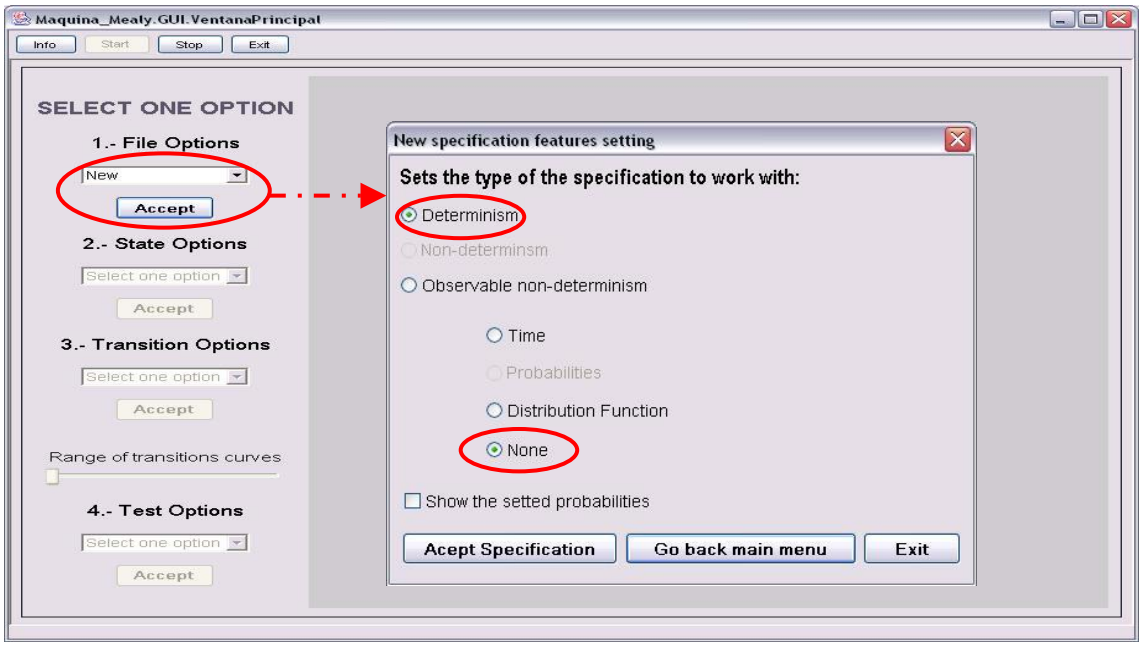
En este apartado se va a explicar cuáles son los pasos necesarios para la creación de una nueva especificación.

Nuestra herramienta está diseñada para distintos marcos formales de actuación, y ofrece por tanto distintas opciones en la creación, permitiendo seleccionar entre distintas característica sen función del objetivo que se pretenda obtener y del estudio que se pretenda realizar.
De esta forma, hay que tener en cuenta que la posterior navegación por la aplicación dependerá del tipo de especificación que se haya decidido implementar.

Muchas de las opciones posteriores a la creación que pueden ejecutarse son comunes a la especificación elegida, por lo que en este manual se basará tan sólo en una en concreto para explicar el funcionamiento de la herramienta, haciendo mención, cuando corresponda, de las novedades y/o diferencias a destacar en caso de que se seleccione otro tipo de especificación.

Concretamente, este manual se basará en la creación de una especificación determinista y sin parámetros no funcionales.

Ahora bien, para crear una nueva especificación, basta con seleccionar del menú “File Options” situado en el panel lateral izquierdo, la opción “New”, y pulsar el botón “Acept” con lo cual se accederá a la ventana “New Specification features setting” en la cual se selecciona el tipo de especificación a crear. Para crear la especificación que servirá de guía a lo largo de este manual, dejar marcadas las opciones que vienen por defecto (Determinism – None) y pulsar sobre el botón “Acept Specificacion” tal y como se muestra a continuación:



Una vez que se hayan concluido estos pasos, se volverá a la ventana principal de la aplicación en la que puede observarse que se han habilitado, por un lado, el menú "State Options", y por otro, el panel de visualización gráfica.

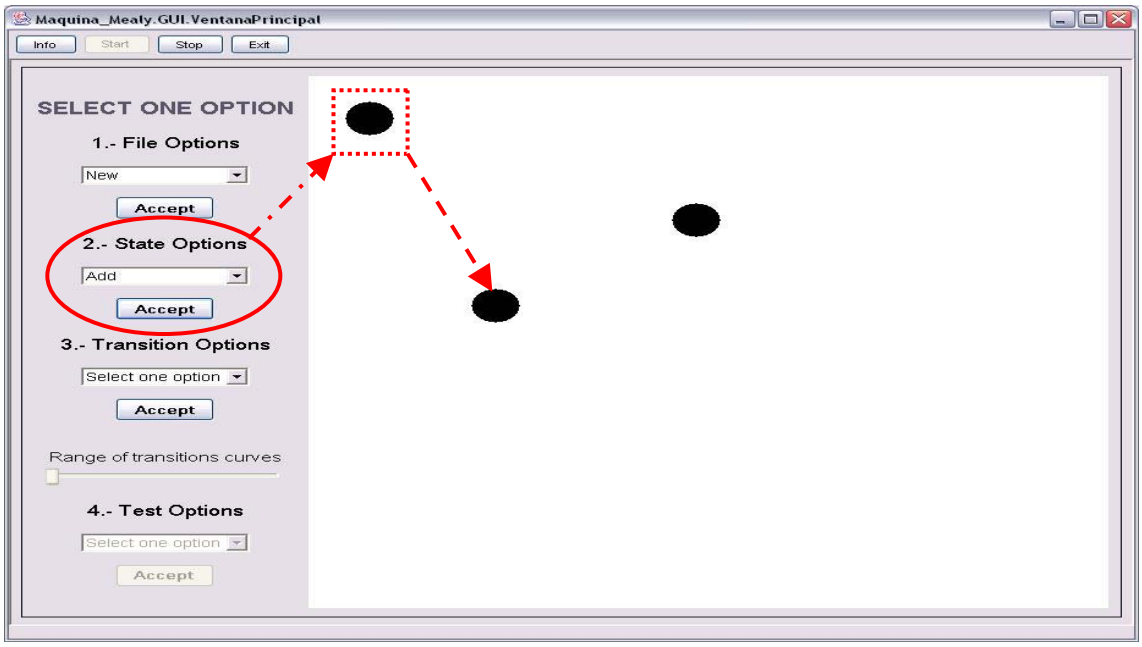
Podremos pasar entonces a añadir los estados que formarán parte de la especificación recién creada.

6.4.2.- Crear estados

Para la adición de estados, basta con seleccionar del menú "State Options" la opción "Add" y pulsar sobre el botón "Acept". Una vez realizado estas acciones, aparecerá en la parte superior izquierda del panel de edición de gráficos una figura que representa el estado recién añadido.

Existe la posibilidad de desplazar dicha figura por el panel para obtener así una distribución de los estados creados que permitan una visualización más clara de la especificación creada por parte del usuario. Para ello basta con mantener pulsado el botón izquierdo del ratón sobre la figura y arrastrarlo hasta la posición deseada.

Dichos pasos se muestran en las imágenes que se exponen a continuación:



Una vez que se ha añadido un estado a la especificación, se habilitará el menú "Transition Options" que permitirá la creación de transiciones como se detalla a continuación.

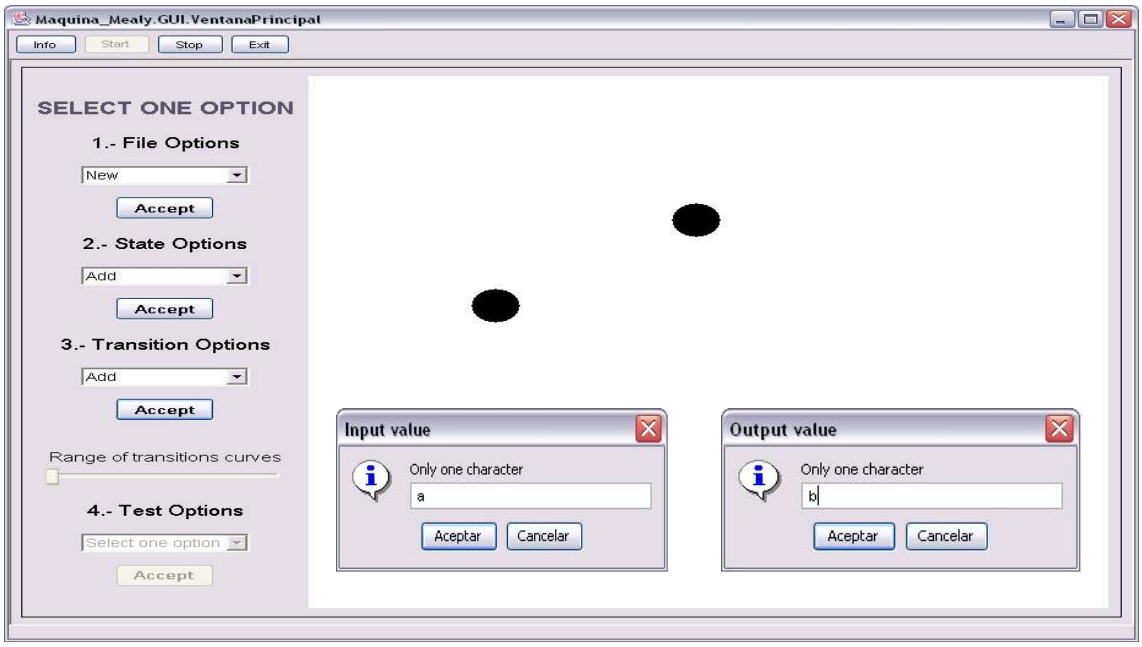
6.4.3.- Crear transiciones

Para añadir una transición basta con seleccionar del menú "Transition Options" la opción "Add" y pulsar sobre el botón "Acept".

Para la especificación que estamos creando, se pedirá al usuario que se introduzcan los valores para el carácter de entrada y de salida de la transición (sólo un carácter está permitido). Una vez introducidos dichos parámetros se tendrá que especificar cuáles van a ser los estados origen y destino de la transición.

Para seleccionar ambos bastará con hacer click sobre los estados que se deseen de entre los que han sido ya añadidos a la especificación. Al seleccionar el estado origen de la transición, el color de éste cambia a rojo para indicar su condición de estado origen. Dicho estado volverá a su color inicial tras seleccionar el estado destino del a transición.

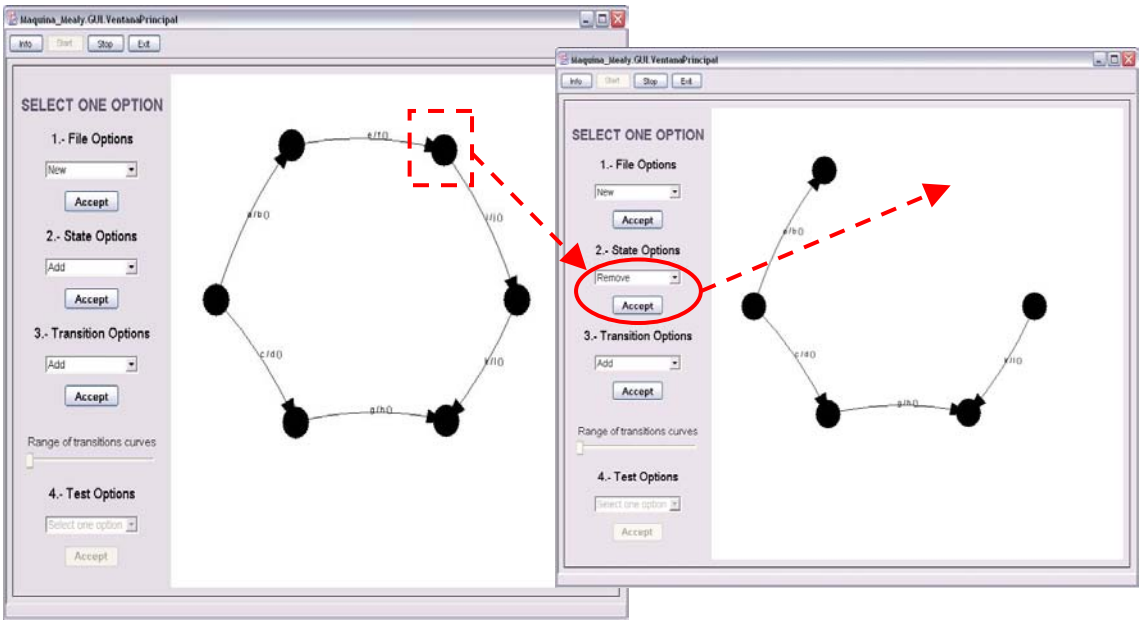
La ejecución de dichas acciones se muestra a continuación:



6.4.4.- Eliminar estados

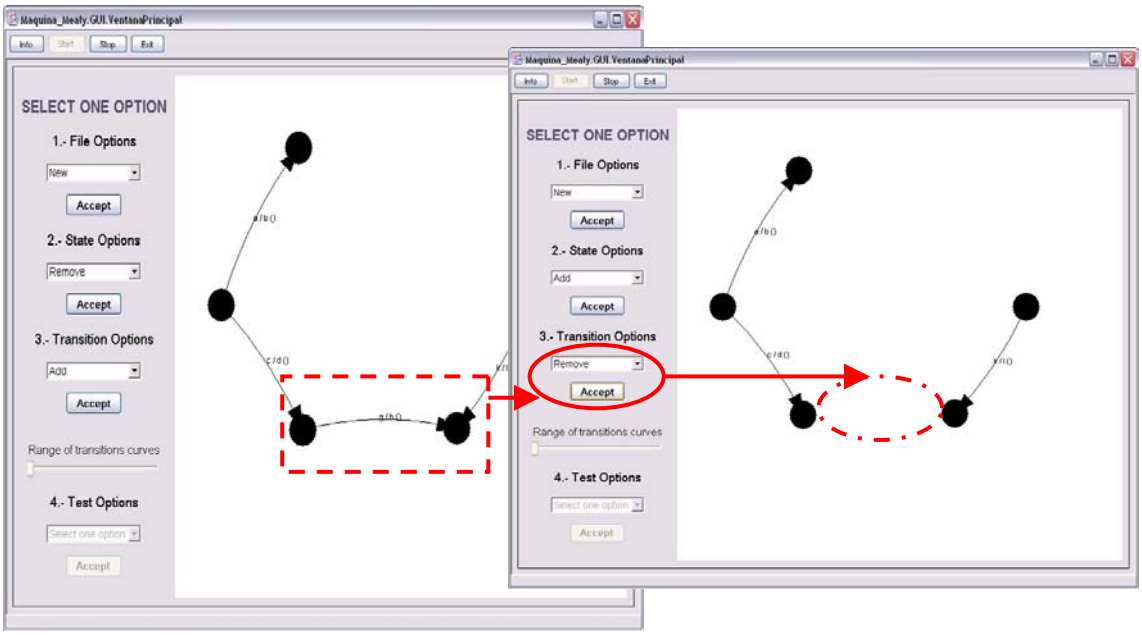
Durante la implementación de la herramienta se ha tenido en cuenta la posibilidad de que el usuario desee eliminar alguno de los estados previamente creados, bien porque hayan sido añadidos por equivocación o bien porque se quiera realizar modificaciones sobre especificaciones existentes para nuevas pruebas.

Para llevar a cabo esta acción, basta con seleccionar del menú “State Options” la opción “Remove” y pulsar el botón “Acept”. A continuación se deberá pulsar sobre el estado que se desea sea eliminado. Hay que tener en cuanto por otro lado, que la eliminación de un estado supone la eliminación de todas sus transiciones, tanto de entrada como de salida, tal y como se muestra a continuación:



6.4.5.- Eliminar transiciones

Para llevar eliminar una transición, basta con seleccionar del menú “Transition Options” la opción “Remove” y pulsar el botón “Acept”. A continuación se deberá pulsar sobre la transición que se desea eliminar, tal y como se muestra a continuación:



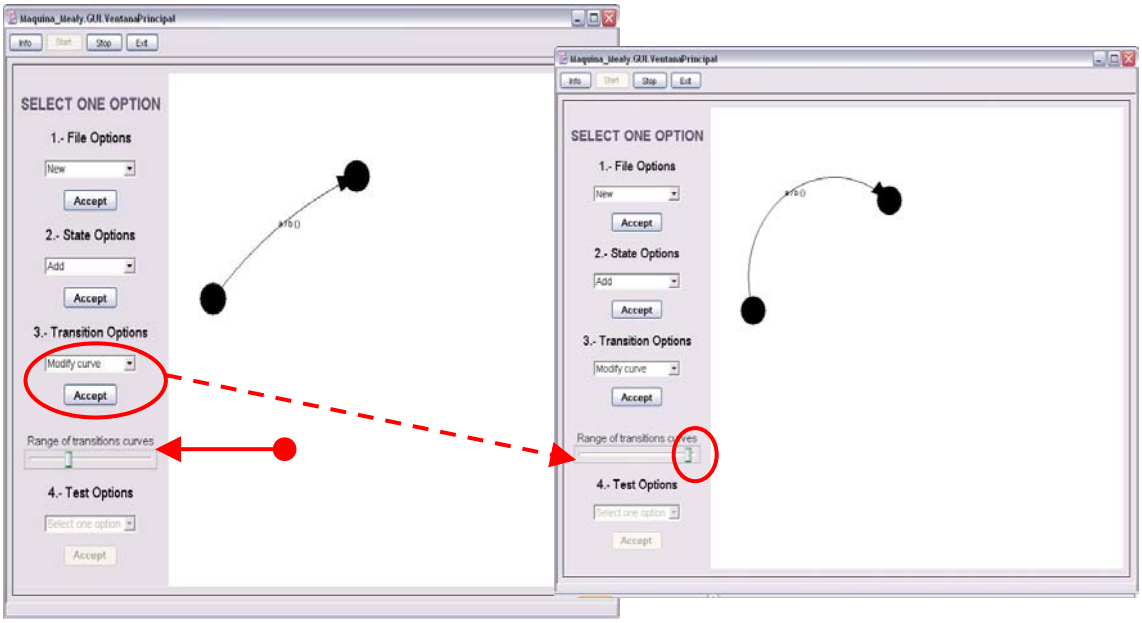
6.4.6.- Modificar curvatura

A lo largo de la realización de la herramienta se ha tenido en cuenta la posibilidad de querer crear transiciones que tengan en común un mismo estado como origen. De hecho, es una situación normal y la mayoría de este tipo de especificaciones funciona de esta manera.

Es por tanto, que se ha implementado una solución para permitir que la visualización de la especificación creada resulte lo más clara posible de cara al usuario. Dicha solución no es otra que permitir modificar la curvatura de las flechas que gráficamente representan a las transiciones.

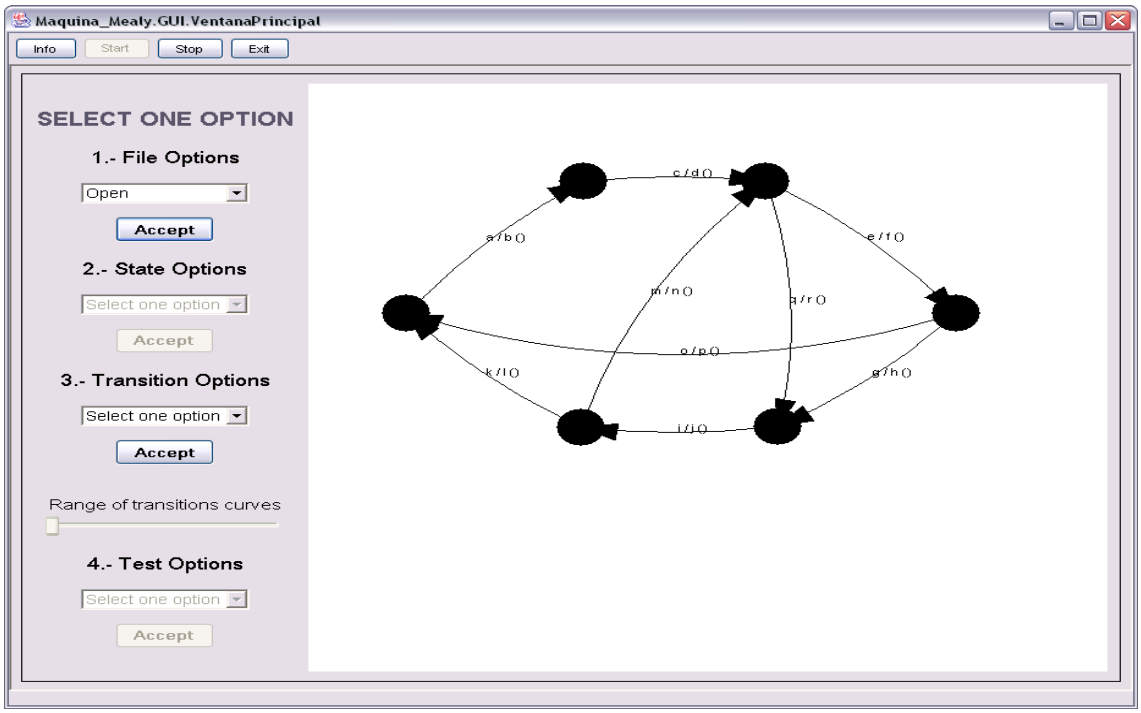
Para realizar esta opción basta con seleccionar del menú "Transition Options" la opción "Modify curve" y pulsar el botón "Acept". Se deberá seleccionar entonces la transición cuya curvatura se quiere modificar y mediante el "Slide" situado en el panel izquierdo, fijar la curvatura que se desee. Podrá verse cómo la transición va modificándose en función del movimiento del citado Slide.

Una muestra de la opción aquí descrita se muestra a continuación:



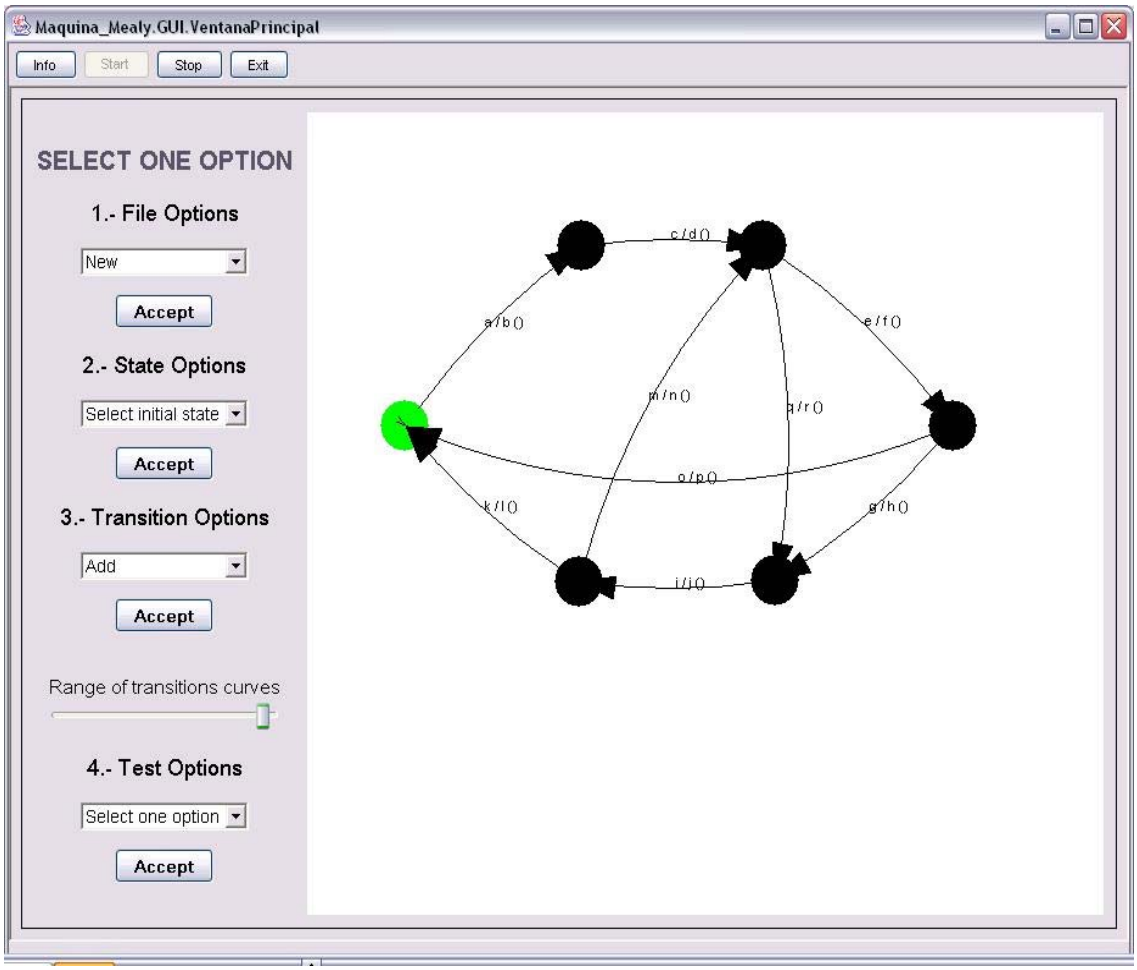
6.4.7.- Seleccionar estado inicial

Si se han seguido todos los pasos hasta ahora explicados se ha disponer ya de una especificación consistente en una serie de estados y sus respectivas transiciones. Un ejemplo de una especificación creada se muestra en la siguiente imagen:



Ahora bien, si lo que se desea a continuación es realizar una derivación de la misma, es requisito imprescindible la selección de un estado inicial. Para ello bastará con seleccionar del menú "State Options" la opción "Select init state", y pulsar el botón "Acept".

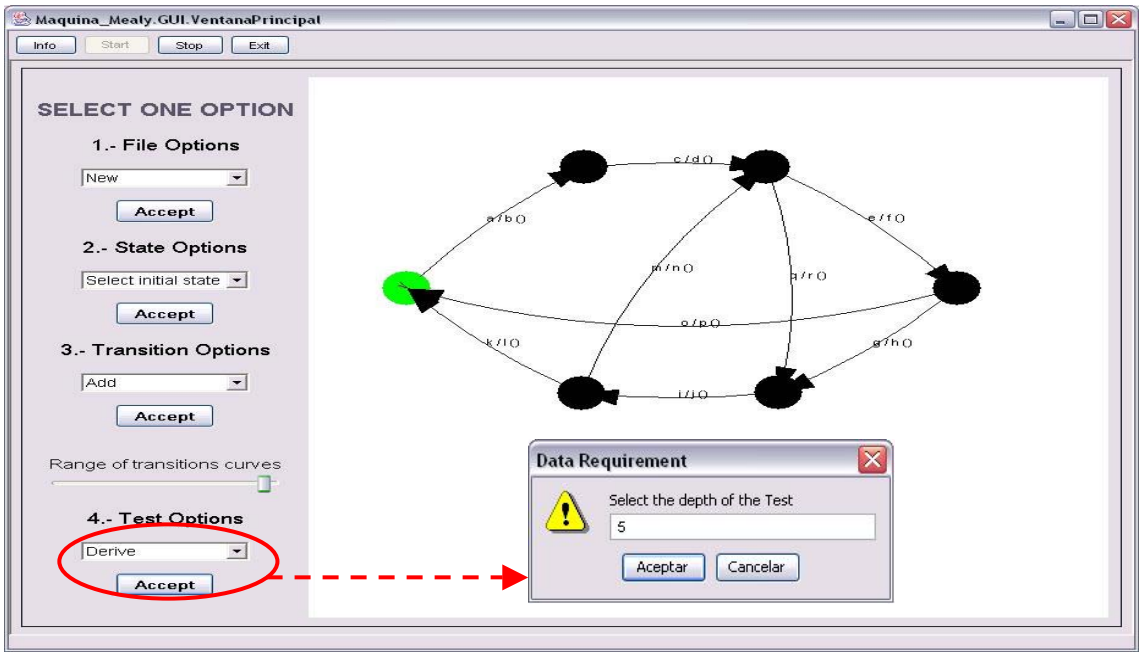
Se deberá seleccionar a continuación, de entre los estados existentes, aquel que va a desempeñar el papel de estado inicial. Bastará con pulsar sobre el estado elegido para desempeñar dicha función. El color del estado seleccionado cambiará a verde como indicativo de su rol, tal y como se muestra a continuación:



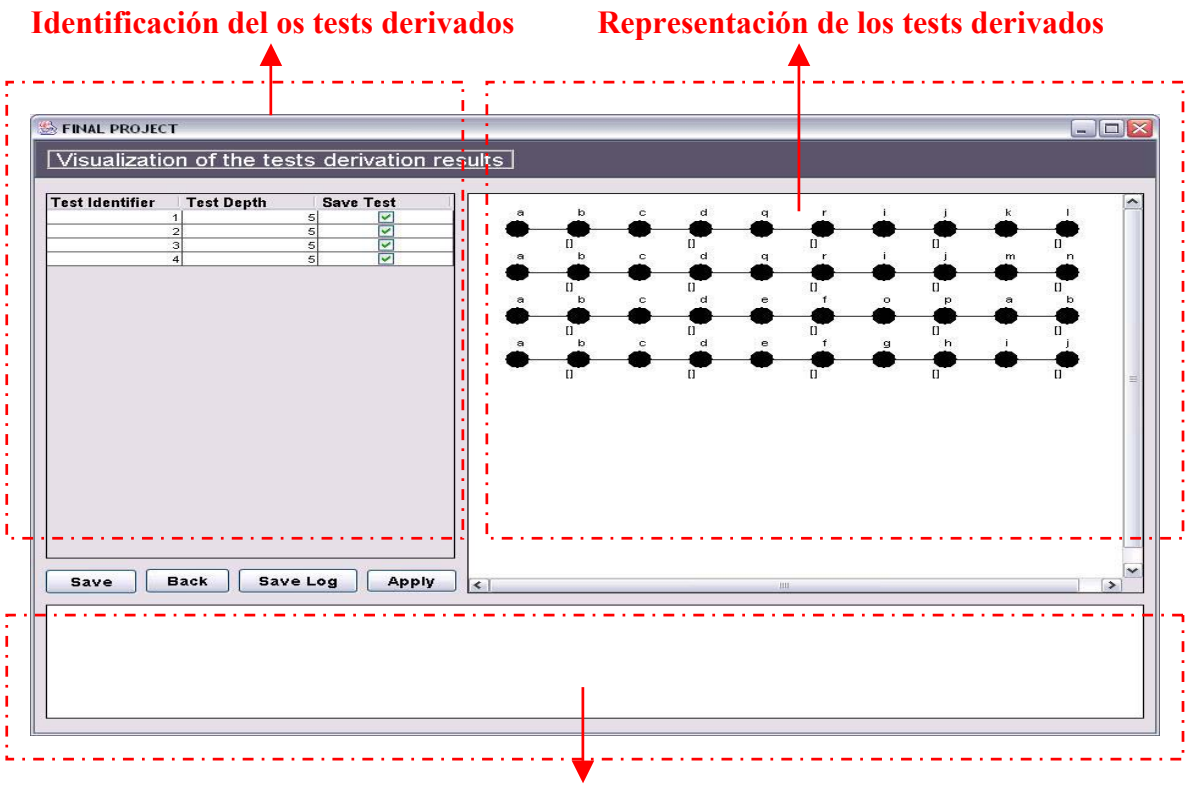
6.4.8.- Derivar test

Una vez creada la especificación, el siguiente paso lógico, es su derivación y posterior visualización de los tests resultantes de dicha derivación. Para mostrar estos tests, basta con seleccionar del menú “Test Options” la opción “Paint” y pulsar el botón “Acept”.

Se pedirá entonces al usuario la introducción del a profundidad máxima del os tests derivados, tal y como se muestra a continuación:



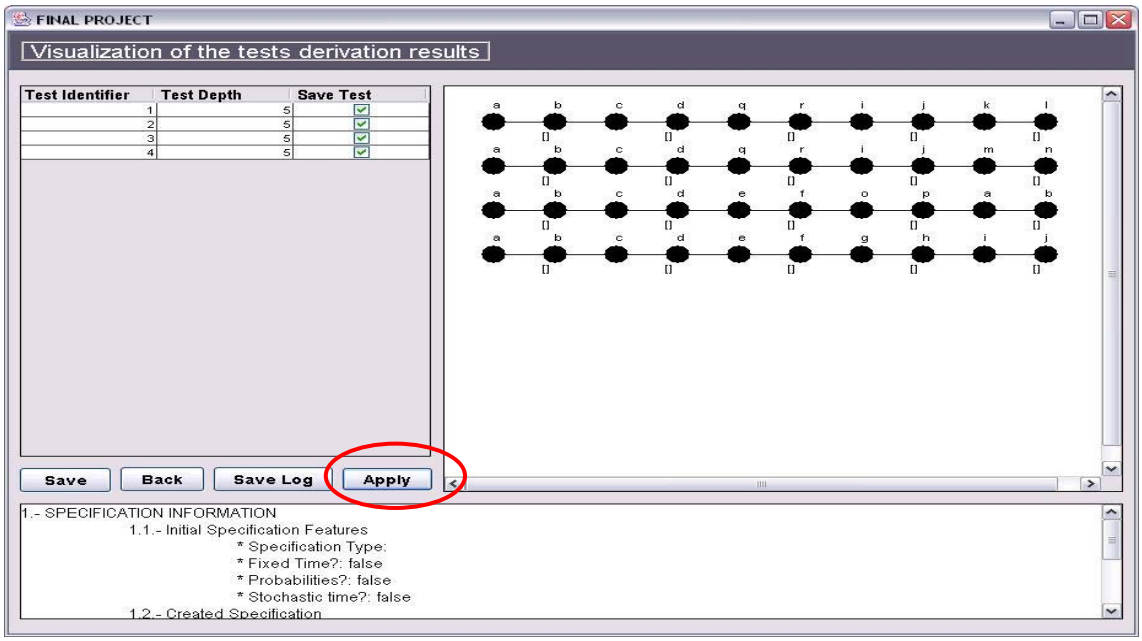
Tras la petición anterior se procederá a la visualización de la siguiente ventana, cuyo contenido se explica en la imagen mostrada a continuación:



6.4.10. Aplicar test a implementación

El objetivo final de la creación de especificaciones y sus correspondientes derivaciones de tests, no es más que la comprobación de si la implementación creada es conforme con la especificación diseñada.

Para ello, y una vez que se han derivado los tests, basta con pulsar sobre el botón "Apply" mostrándose en el panel de visualización del log el resultado obtenido, tal y como se muestra a continuación:



6.4.11. Salvado de tests y log

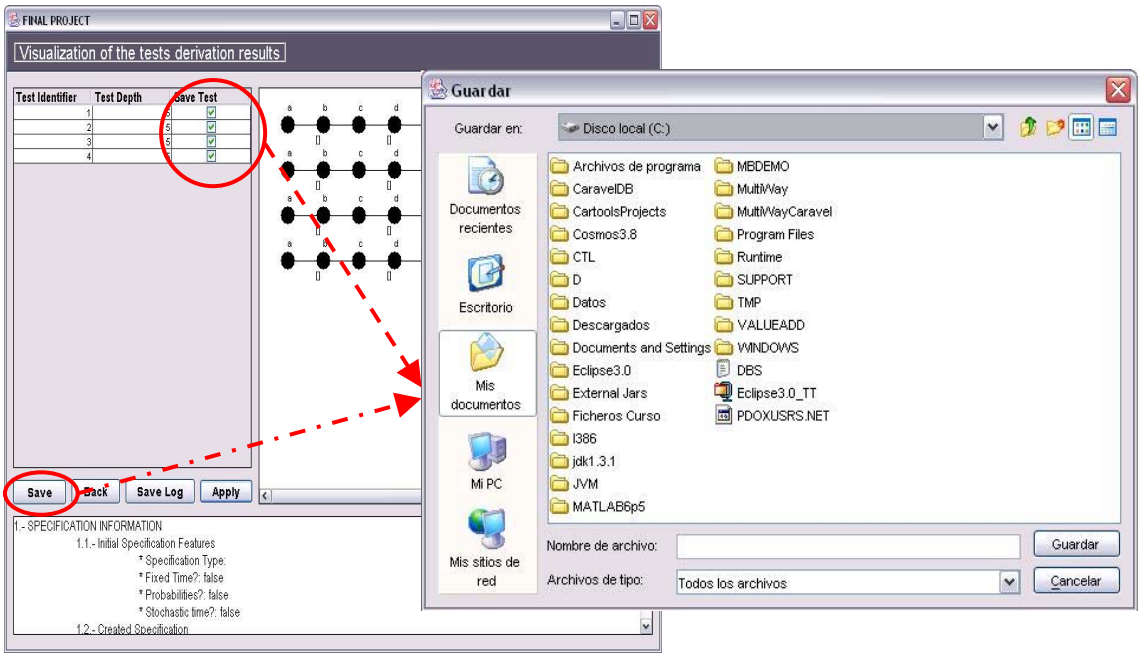
Puede resultar interesante el guardar los tests derivados correspondientes a una especificación concreta así como el resultado obtenido tras su aplicación a la implementación correspondiente.

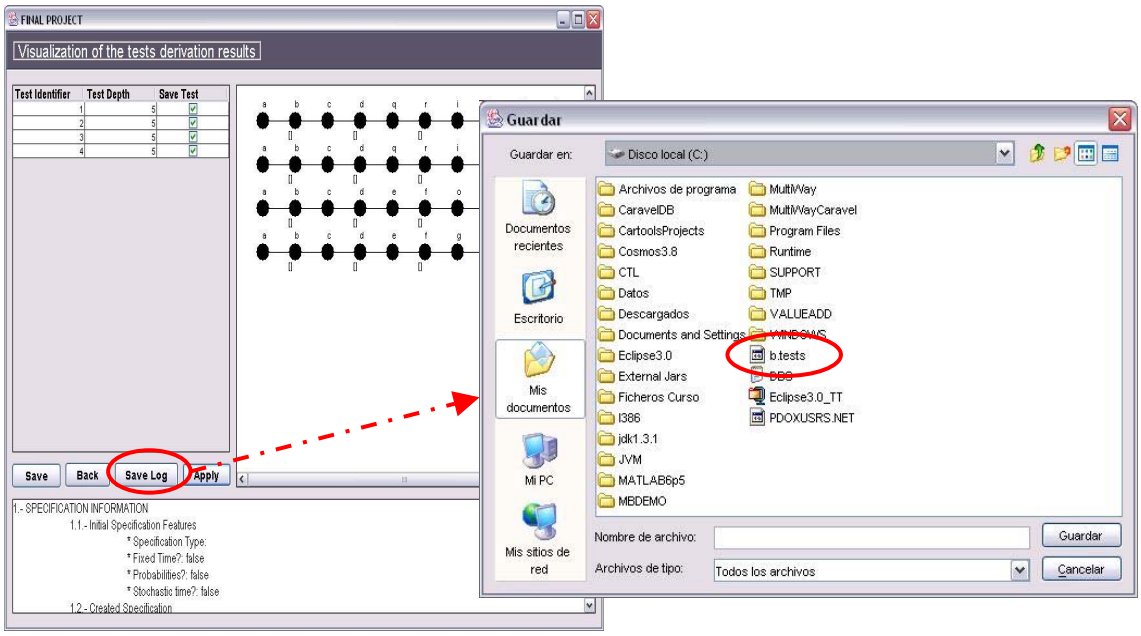
Para ello basta con pulsar sobre el botón "Save", en el caso de querer salvar los tests derivados, y tras haber seleccionado previamente cuáles de los tests obtenidos se desean guardar. En principio se presupone que se desea guardar todos los tests que han sido derivados, de ahí el hecho de que todas las casillas de la tabla correspondiente a esta

condición están marcadas inicialmente. Se podrá por tanto, seleccionar qué tests no se desea guardar deseleccionando dicha casilla para el test correspondiente.

En cuanto al salvado del log obtenido, tan solo es necesario pulsar sobre el botón “Save log” y tal y como ocurre en el caso de los tests, seleccionar posteriormente el directorio donde sea desea guardar.

Dichas acciones se muestran a continuación:





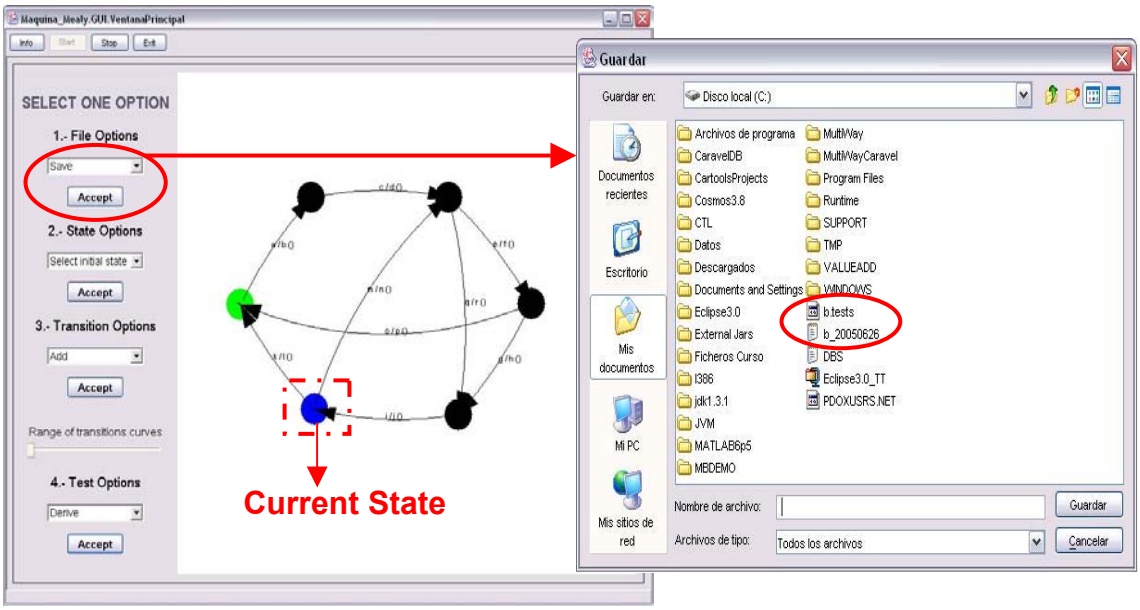
Es importante tener en cuenta que la extensión de los tests salvados es, por defecto, ".test", y por otro lado, la extensión por defecto del log salvado, es ".log". No obstante estas extensiones son variables y orientativas por lo que el cambio de las mismas no supone un obstáculo en el correcto funcionamiento del a aplicación.

6.4.12.- Salvar / Abrir especificaciones

Como una opción más, se ofrece al usuario la posibilidad de salvar las especificaciones previamente creadas, así como su posterior recuperación para tenerlas nuevamente disponibles para futuras modificaciones o nuevas pruebas sobre las mismas.

Para salvar una especificación basta con seleccionar del menú "File Options" la opción "Save" y pulsar el botón "Acept". De esta forma se mostrará el cuadro de diálogo correspondiente para el almacenamiento de la especificación en curso, con una extensión por defecto de ".spec".

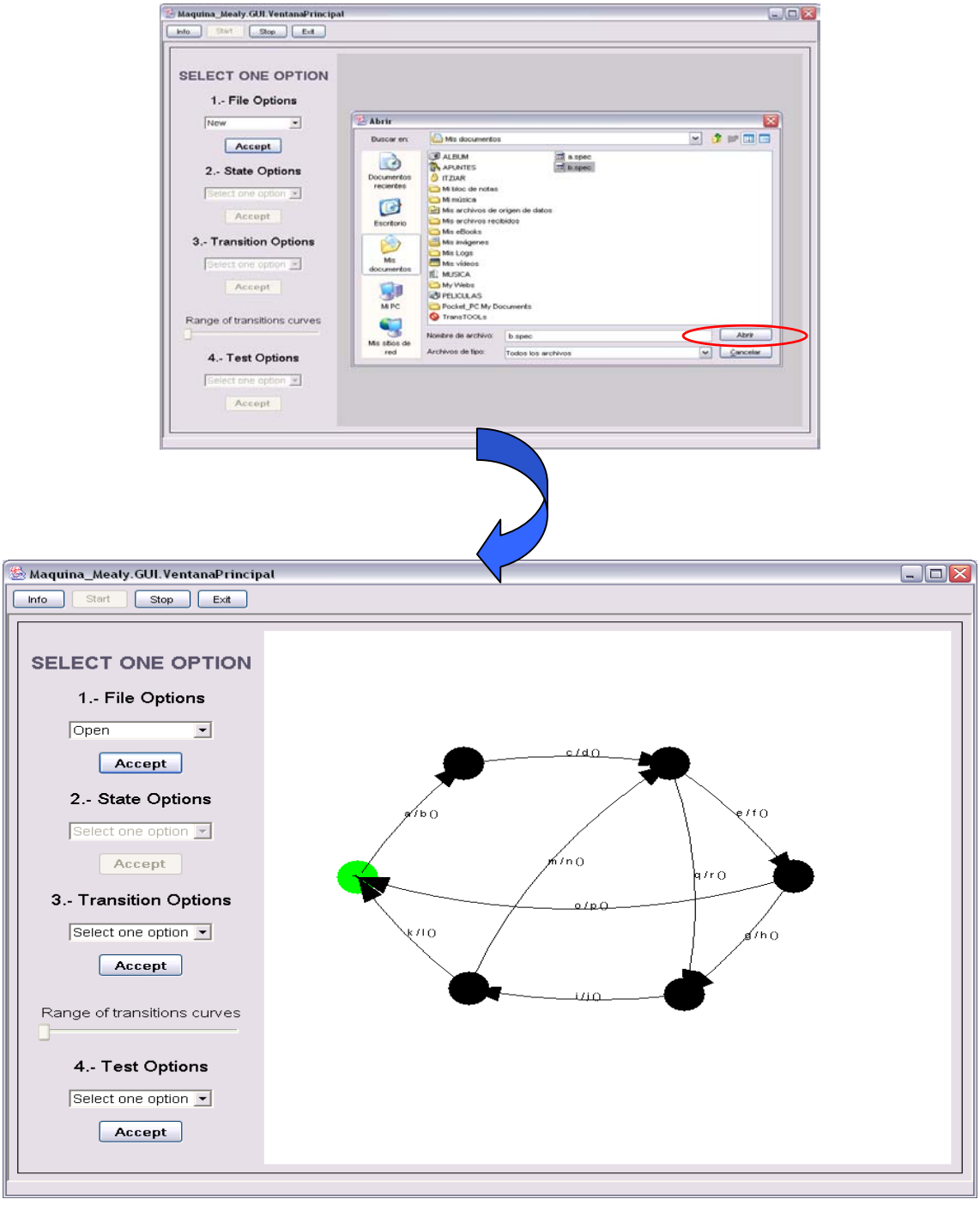
Dicha acción se muestra a continuación:



Así mismo, y una vez almacenada la especificación, ésta podrá ser recuperada posteriormente para su visualización y edición si así se desea. Para ello bastará con

seleccionar del menú “File Options” la opción “Open” y pulsar el botón “Acept”, tras lo cual se mostrará el cuadro de diálogo necesario para la selección del a especificación a ser abierta.

Una vez seleccionada dicha especificación, ésta será de nuevo mostrada en el panel de representación gráfica de la ventana principal y estará disponible para su edición si así se desea, tal y como se muestra a continuación:



6.5.- CONSIDERACIONES

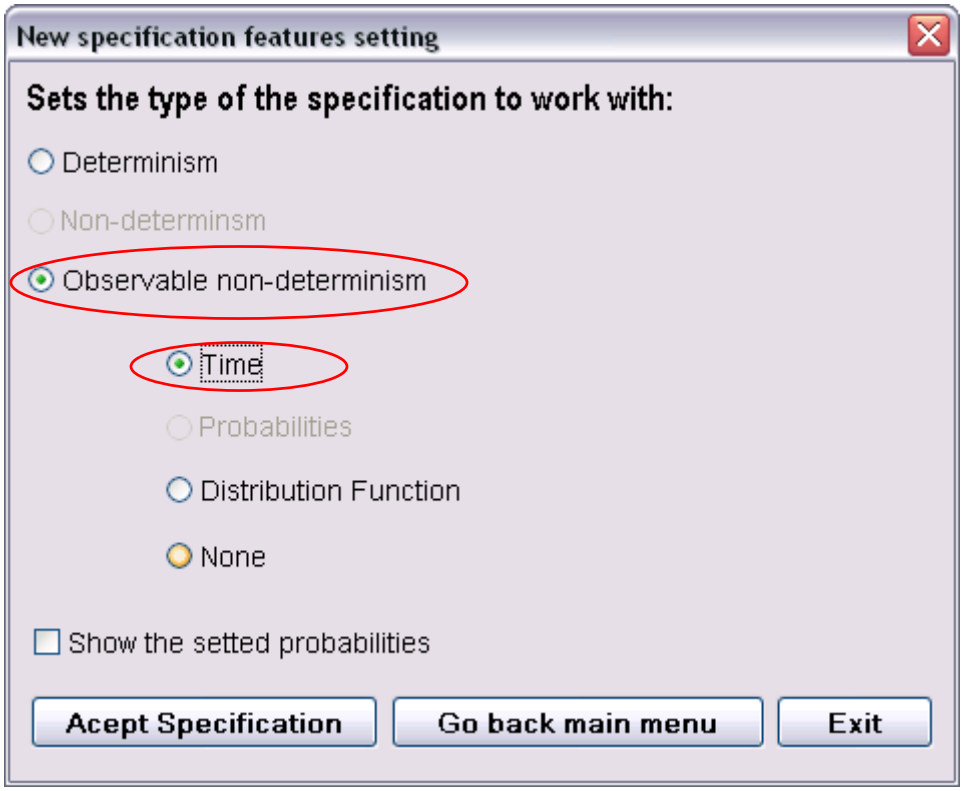
Como se ha venido explicando a lo largo de este manual, la herramienta aquí presentada ofrece la posibilidad de disponer entre diferentes marcos formales de actuación.

Para el desarrollo de este manual nos hemos basado en la referida a especificaciones de propiedades funcionales. Para el resto del tipo de especificaciones disponibles el funcionamiento es el mismo, con la salvedad de la adición del atributo tiempo a la especificación.

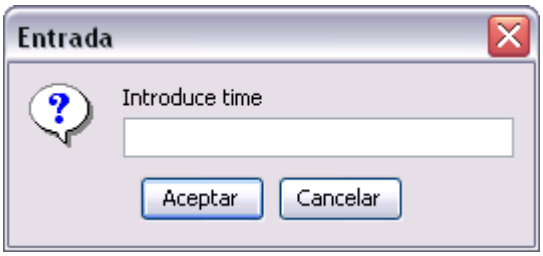
Vamos por tanto, en este apartado a exponer qué diferencias específicas nos encontramos en la creación de especificaciones de propiedades no funcionales.

6.5.1.- Sistema temporizador

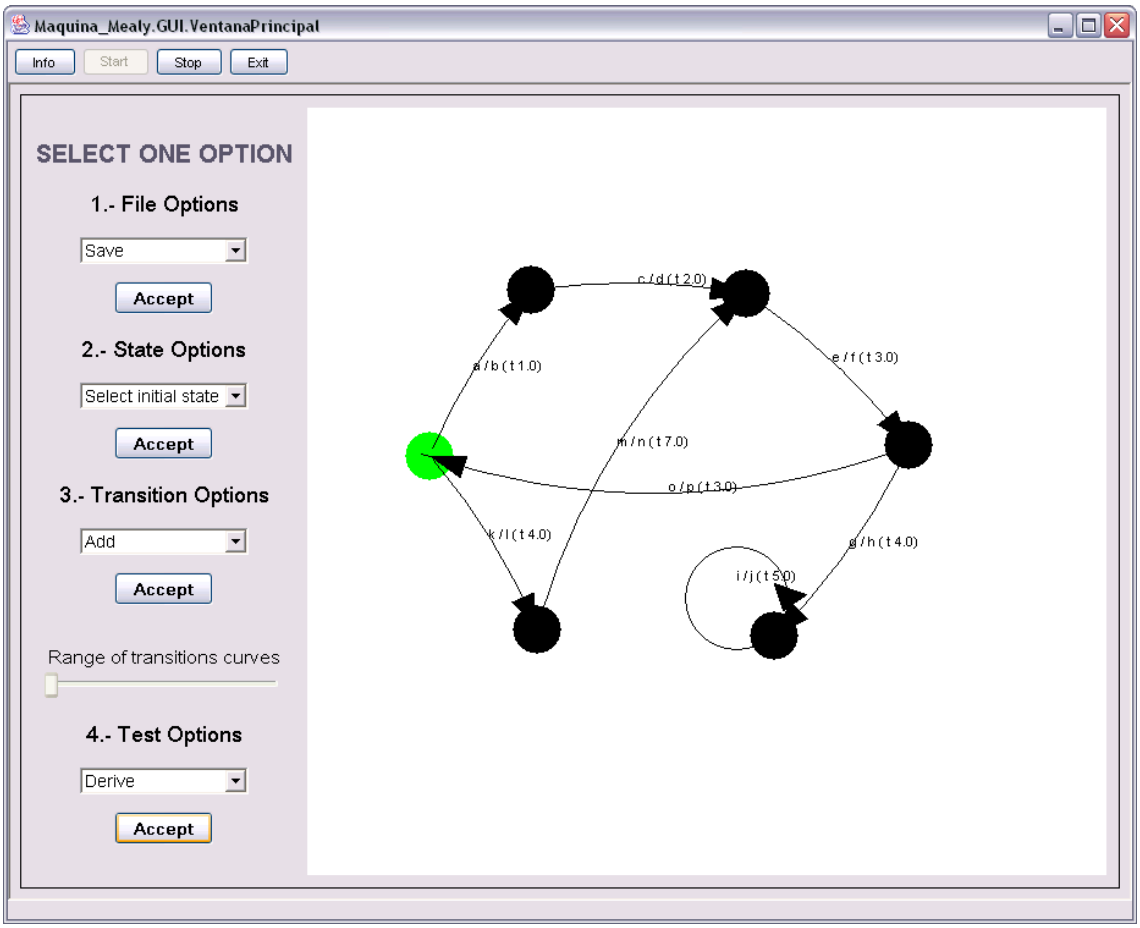
Para diseñar un sistema de estas características basta con seleccionar, cuando sec rea la especificación, las siguientes características:



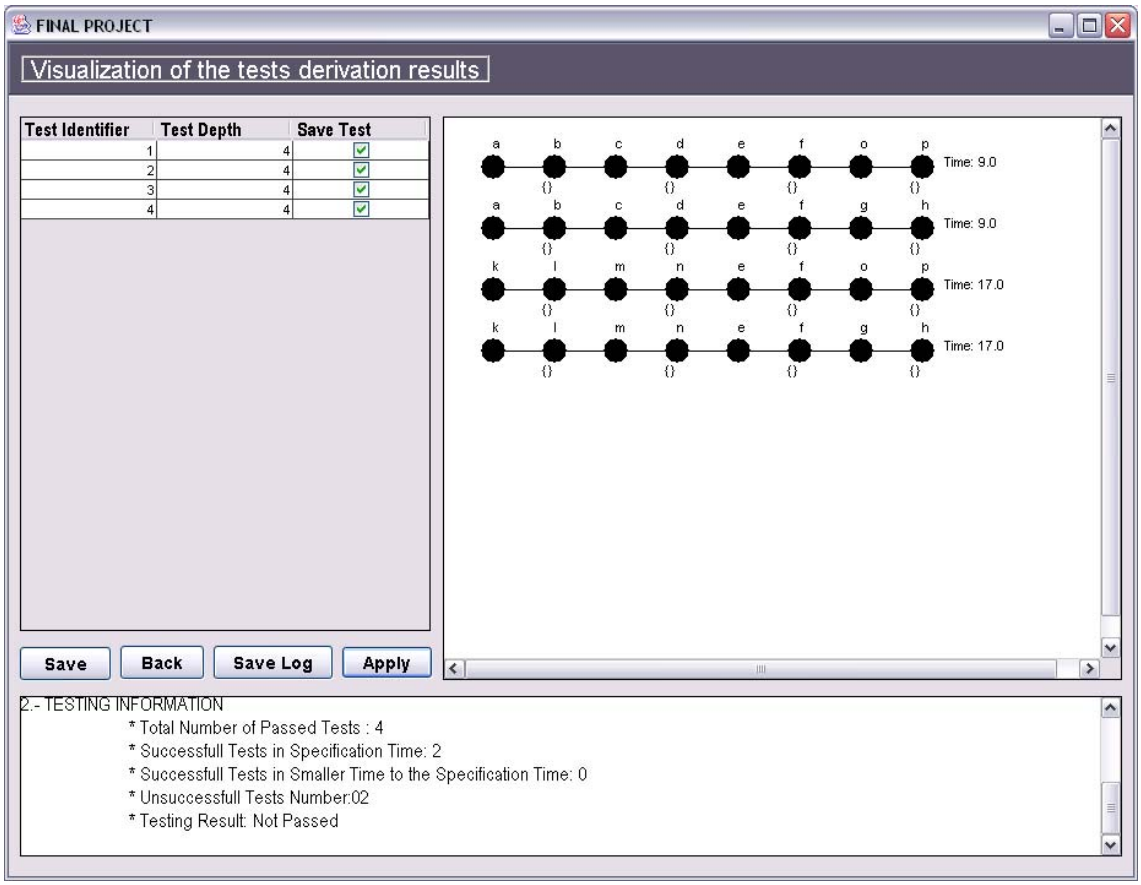
Ahora bien, a continuación hay que tener en cuenta que cuando se añadan transiciones a la especificación, se pedirá al usuario que introduzca un valor para la transición que se está creando. Esta petición se realiza mediante una ventana del siguiente tipo:



Se muestra a continuación como quedaría la representación del a especificación con la introducción de tiempos, así como el resultado del a derivación de tests:

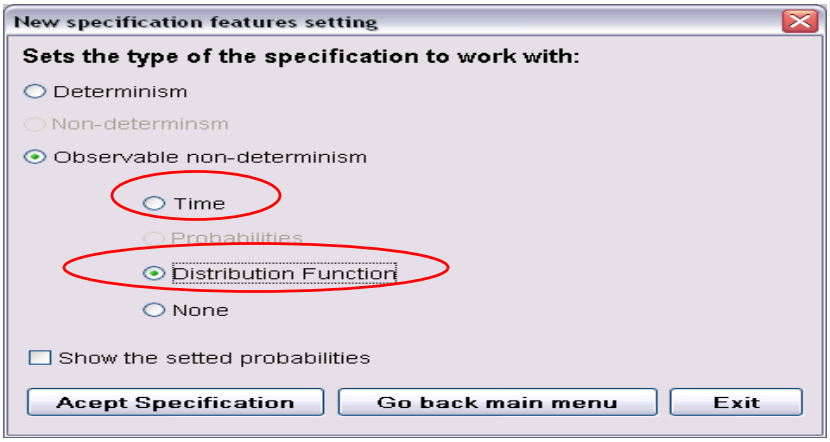


Y tras la derivación de los tests, obtenemos:

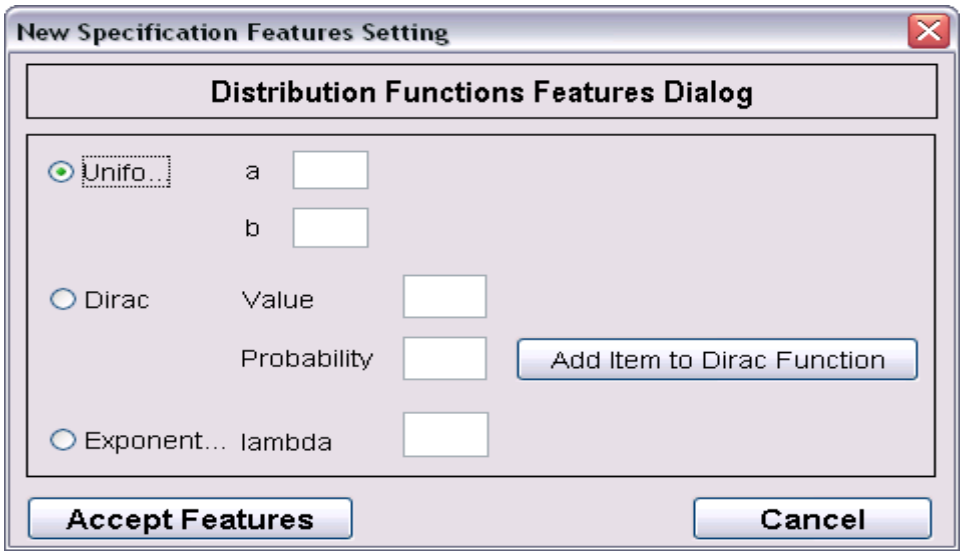


6.5.2.- Sistema con tiempos estocásticos

Para diseñar un sistema de estas características basta con seleccionar, cuando sec rea la especificación, las siguientes características:

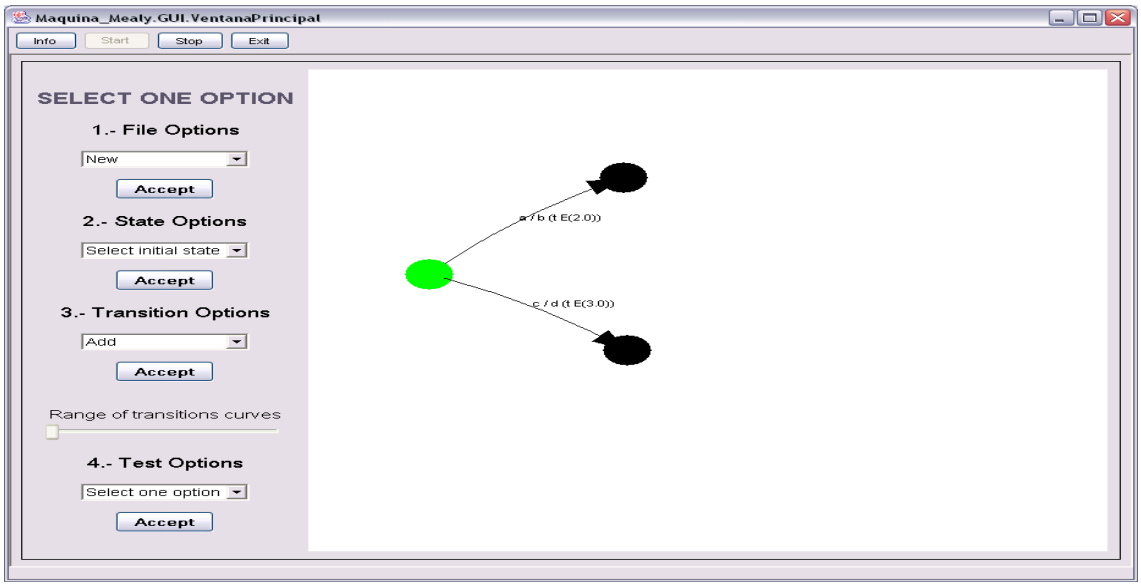


Ahora bien, a continuación hay que tener en cuenta que cuando se añadan transiciones a la especificación, se pedirá al usuario que introduzca un valor para la transición que se está creando. Esta petición se realiza mediante una ventana del siguiente tipo:

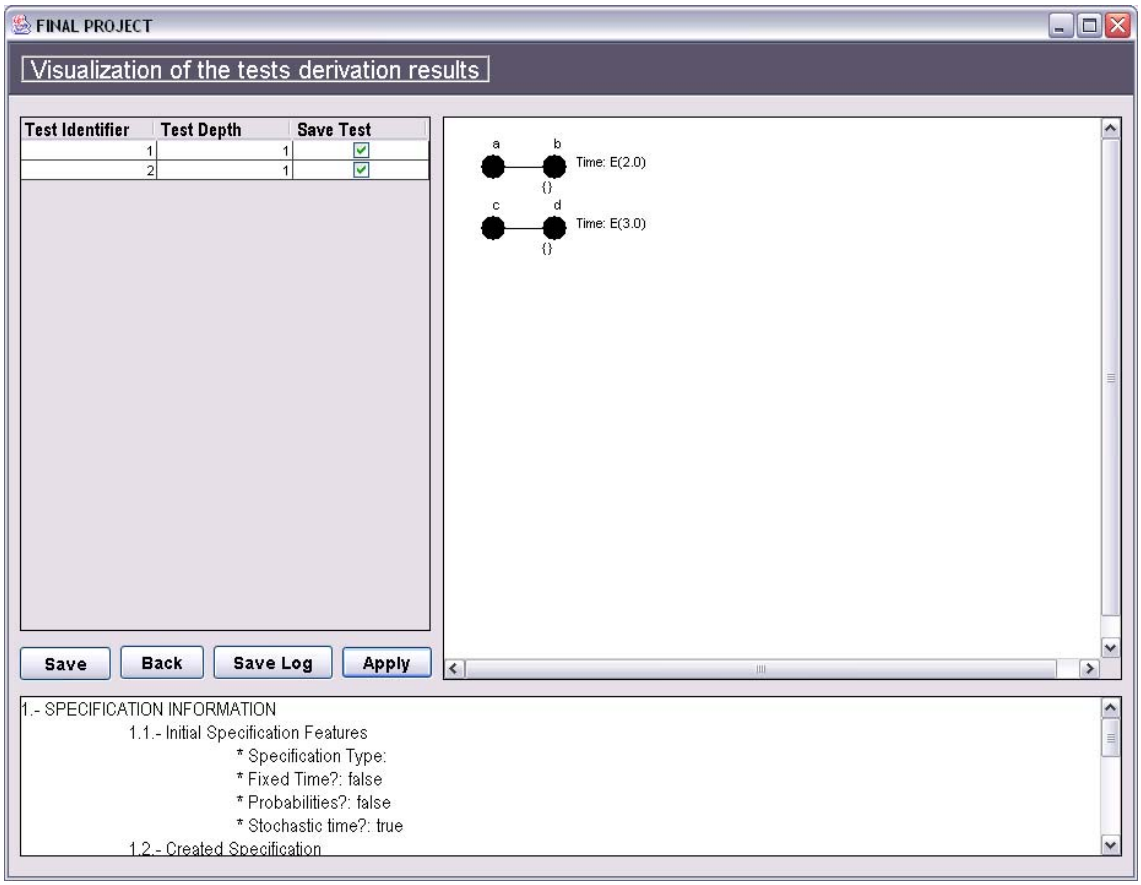


En la que se podrá determinar que tipo de función de distribución se va a utilizar para la generación del tiempo.

Se muestra a continuación como quedaría la representación del a especificación con la introducción de tiempos, así como el resultado del a derivación de tests:



Y tras la derivación de los tests, obtenemos:



Como ha podido observarse, el funcionamiento del a herramienta es prácticamente idéntica independientemente de la especificación creada.

7.- BIBLIOGRAFÍA

- **Artículo "Towards Testing Stochastic Timed Systems"**

  Manuel Núñez and Ismael Rodríguez
  Dept. Sistemas Informáticos y Programación, Facultad de Informática,
  Universidad Complutense de Madrid, E-28040 Madrid, Spain,
  {mn,isrodrig}@sip.ucm.es

- **Artículo "Conformance Testing Relations for Timed Systems"**

  Dept. Sistemas Informáticos y Programación, Facultad de Informática,
  Universidad Complutense de Madrid, E-28040 Madrid, Spain,
  {mn,isrodrig}@sip.ucm.es